305-CD-044-001

EOSDIS Core System Project

Flight Operations Segment (FOS) Telemetry Design Specification for the ECS Project

October 1995

Hughes Information Technology Corporation
Upper Marlboro, MD

Flight Operations Segment (FOS) Telemetry Design Specification for the ECS Project

October 1995

Prepared Under Contract NAS5-60000
CDRL Item #046

APPROVED BY

Cal Moore /s/ 9/22/95

Cal Moore, FOS CCB Chairman Date
EOSDIS Core System Project

Hughes Information Technology Corporation
Upper Marlboro, Maryland

This page intentionally left blank.

Preface

This document, one of nineteen, comprises the detailed design specification of the FOS subsystems for Releases A and B of the ECS project. This includes the FOS design to support the AM-1 launch.

The FOS subsystem design specification documents for Releases A and B of the ECS project include:

305-CD-040	FOS Design Specification (Segment Level Design)
305-CD-041	Planning and Scheduling Design Specification
305-CD-042	Command Management Design Specification
305-CD-043	Resource Management Design Specification
305-CD-044	Telemetry Design Specification
305-CD-045	Command Design Specification
305-CD-046	Real-Time Contact Management Design Specification
305-CD-047	Analysis Design Specification
305-CD-048	User Interface Design Specification
305-CD-049	Data Management Design Specification
305-CD-050	Planning and Scheduling Program PDL
305-CD-051	Command Management PDL
305-CD-052	Resource Management PDL
305-CD-053	Telemetry PDL
305-CD-054	Real-Time Contact Management PDL
305-CD-055	Analysis PDL
305-CD-056	User Interface PDL
305-CD-057	Data Management PDL
305-CD-058	Command PDL

Object models presented in this document have been exported directly from CASE tools and in some cases contain too much detail to be easily readable within hard copy page constraints. The reader is encouraged to view these drawings on line using the Portable Document Format (PDF) electronic copy available via the ECS Data Handling System (EDHS) at URL http://edhs1.gsfc.nasa.gov.

This document is a contract deliverable with an approval code 2. As such, it does not require formal Government approval, however, the Government reserves the right to request changes within 45 days of the initial submittal. Once approved, contractor changes to this document are handled in accordance with Class I and Class II change control requirements described in the EOS Configuration Management Plan, and changes to this document shall be made by document change notice (DCN) or by complete revision.

Any questions should be addressed to:

Data Management Office
The ECS Project Office
Hughes Information Technology Corporation
1616 McCormick Drive
Upper Marlboro, MD 20785

Abstract

The FOS Design Specification consists of a set of 19 documents that define the FOS detailed design. The first document, the FOS Segment Level Design, provides an overview of the FOS segment design, the architecture, and analyses and trades. The next nine documents provide the detailed design for each of the nine FOS subsystems. The last nine documents provide the PDL for the nine FOS subsystems. It also allocates the level 4 FOS requirements to the subsystem design.

Keywords: FOS, design, specification, analysis, IST, EOC

This page intentionally left blank.

Change Information Page

List of Effective Pages	
Page Number	**Issue**
Title	Original
iii through vii	Original
1 -1 and 1-2	Original
2-1 through 2-4	Original
3-1 through 3-84	Original
GL-1 through GL-8	Original

Document History			
Document Number	**Status/Issue**	**Publication Date**	**CCR Number**
305-CD-044-001	Original	October 1995	95-0679

This page intentionally left blank.

Contents

Preface

Abstract

Change Information Page

1. Introduction

2. Related Documentation

3.Telemetry

Abbreviations and Acronyms

Glossary

Figures

Tables

Abbreviations and Acronyms

Glossary

This page intentionally left blank.

1. Introduction

1.1 Identification

The contents of this document defines the design specification for the Flight Operations Segment (FOS). Thus, this document addresses the Data Item Description (DID) for CDRL item 046 305/DV2 under Contract NAS5-60000.

1.2 Scope

The Flight Operations Segment (FOS) Design Specification defines the detailed design of the FOS. It allocates the level 4 FOS requirements to the subsystem design. It also defines the FOS architectural design. In particular, this document addresses the Data Item Description (DID) for CDRL # 053, the Segment Design Specification.

This document reflects the August 23, 1995 Technical Baseline maintained by the contractor configuration control board in accordance with ECS Technical Direction No. 11, dated December 6, 1994. It covers releases A and B for FOS. This corresponds to the design to support the AM-1 launch.

1.3 Purpose

The FOS Design Specification consists of a set of 19 documents that define the FOS detailed design. The first document, the FOS Segment Level Design, provides an overview of the FOS segment design, the architecture, and analyses and trades. The next nine documents provide the detailed design for each of the nine FOS subsystems. The last nine documents provide the PDL for the nine FOS subsystems.

1.4 Status and Schedule

This submittal of DID 305/DV2 incorporates the FOS detailed design performed during the Critical Design Review (CDR) time frame. This document is under the ECS Project configuration control.

1.5 Document Organization

305-CD-040 contains the overview, the FOS segment models, the FOS architecture, and FOS analyses and trades performed during the design phase.

305-CD-041 contains the detailed design for Planning and Scheduling Design Specification.

305-CD-042 contains the detailed design for Command Management Design Specification.

305-CD-043 contains the detailed design for Resource Management Design Specification.

305-CD-044 contains the detailed design for Telemetry Design Specification.

305-CD-045 contains the detailed design for Command Design Specification.

305-CD-046 contains the detailed design for Real-Time Contact Management Design Specification.

305-CD-047 contains the detailed design for Analysis Design Specification.

305-CD-048 contains the detailed design for User Interface Design Specification.

305-CD-049 contains the detailed design for Data Management Design Specification.

305-CD-050 contains Planning and Scheduling PDL.

305-CD-051 contains Command Management PDL.

305-CD-052 contains Resource Management PDL.

305-CD-053 contains the Telemetry PDL.

305-CD-054 contains the Real-Time Contact Management PDL.

305-CD-055 contains the Analysis PDL.

305-CD-056 contains the User Interface PDL.

305-CD-057 contains the Data Management PDL.

305-CD-058 contains the Command PDL.

Appendix A of the first document contains the traceability between Level 4 Requirements and the design. The traceability maps the Level 4 requirements to the objects included in the subsystem object models.

Glossary contains the key terms that are included within this design specification.

Abbreviations and acronyms contains an alphabetized list of the definitions for abbreviations and acronyms used within this design specification.

2. Related Documentation

2.1 Parent Document

The parent documents are the documents from which this FOS Design Specification's scope and content are derived.

194-207-SE1-001	System Design Specification for the ECS Project
304-CD-001-002	Flight Operations Segment (FOS) Requirements Specification for the ECS Project, Volume 1: General Requirements
304-CD-004-002	Flight Operations Segment (FOS) Requirements Specification for the ECS Project, Volume 2: Mission Specific

2.2 Applicable Documents

The following documents are referenced within this FOS Design Specification or are directly applicable, or contain policies or other directive matters that are binding upon the content of this volume.

194-219-SE1-020	Interface Requirements Document Between EOSDIS Core System (ECS) and NASA Institutional Support Systems
209-CD-002-002	Interface Control Document Between EOSDIS Core System (ECS) and ASTER Ground Data System, Preliminary
209-CD-003-002	Interface Control Document Between EOSDIS Core System (ECS) and the EOS-AM Project for AM-1 Spacecraft Analysis Software, Preliminary
209-CD-004-002	Data Format Control Document for the Earth Observing System (EOS) AM-1 Project Data Base, Preliminary
209-CD-025-001	ICD Between ECS and AM1 Project Spacecraft Software Development and Validation Facilities (SDVF)
311-CD-001-003	Flight Operations Segment (FOS) Database Design and Database Schema for the ECS Project
502-ICD-JPL/GSFC	Goddard Space Flight Center/MO&DSD, Interface Control Document Between the Jet Propulsion Laboratory and the Goddard Space Flight Center for GSFC Missions Using the Deep Space Network
530-ICD-NCCDS/MOC	Goddard Space Flight Center/MO&DSD, Interface Control Document Between the Goddard Space Flight Center Mission Operations Centers and the Network Control Center Data System
530-ICD-NCCDS/POCC	Goddard Space Flight Center/MO&DSD, Interface Control Document Between the Goddard Space Flight Center Payload Operations Control Centers and the Network Control Center Data System

530-DFCD-NCCDS/POCC	Goddard Space Flight Center/MO&DSD, Data Format control Document Between the Goddard Space Flight Center Payload Operations Control Centers and the Network Control Center Data System
540-041	Interface Control Document (ICD) Between the Earth Observing System (EOS) Communications (Ecom) and the EOS Operations Center (EOC), Review
560-EDOS-0230.0001	Goddard Space Flight Center/MO&DSD, Earth Observing System (EOS) Data and Operations System (EDOS) Data Format Requirements Document (DFRD)
ICD-106	Martin Marietta Corporation, Interface Control Document (ICD) Data Format Control Book for EOS-AM Spacecraft
none	Goddard Space Flight Center, Earth Observing System (EOS) AM-1 Flight Dynamics Facility (FDF) / EOS Operations Center (EOC) Interface Control Document

2.3 Information Documents

2.3.1 Information Document Referenced

The following documents are referenced herein and, amplify or clarify the information presented in this document. These documents are not binding on the content of this FOS Design Specification.

194-201-SE1-001	Systems Engineering Plan for the ECS Project
194-202-SE1-001	Standards and Procedures for the ECS Project
193-208-SE1-001	Methodology for Definition of External Interfaces for the ECS Project
308-CD-001-004	Software Development Plan for the ECS Project
194-501-PA1-001	Performance Assurance Implementation Plan for the ECS Project
194-502-PA1-001	Contractor's Practices & Procedures Referenced in the PAIP for the ECS Project
604-CD-001-004	Operations Concept for the ECS Project: Part 1-- ECS Overview, 6/95
604-CD-002-001	Operations Concept for the ECS project: Part 2B -- ECS Release B, Annotated Outline, 3/95
604-CD-003-001	ECS Operations Concept for the ECS Project: Part 2A -- ECS Release A, Final, 7/95
194-WP-912-001	EOC/ICC Trade Study Report for the ECS Project, Working Paper
194-WP-913-003	User Environment Definition for the ECS Project, Working Paper
194-WP-920-001	An Evaluation of OASIS-CC for Use in the FOS, Working Paper
194-TP-285-001	ECS Glossary of Terms
222-TP-003-006	Release Plan Content Description

none	Hughes Information Technology Company, Technical Proposal for the EOSDIS Core System (ECS), Best and Final Offer
560-EDOS-0211.0001	Goddard Space Flight Center, Interface Requirements Document (IRD) Between the Earth Observing System (EOS) Data and Operations System (EDOS), and the EOS Ground System (EGS) Elements, Preliminary
NHB 2410.9A	NASA Hand Book: Security, Logistics and Industry Relations Division, NASA Security Office: Automated Information Security Handbook

This page intentionally left blank.

3. Telemetry

The telemetry subsystem provides the capability to ingest, decommutate, convert, and limit check housekeeping, memory dump, or engineering telemetry data from the EOS spacecraft and instruments. The telemetry subsystem also provides mechanisms to calculate derived parameters and to extract and forward subsets of the processed telemetry. The telemetry subsystem has the ability to receive and process real-time contact or historical replay telemetry. Real-time spacecraft state-checking is an additional capability.

3.1 Telemetry Context Description

The telemetry subsystem context diagram shown in Figure 3.1-1 depicts the data flows between the FOS Telemetry Subsystem and the external ground system as well as EOC internal components. Descriptions of the data flows are summarized for each component:

EDOS: The EDOS forwards telemetry to the Telemetry Subsystem via EDOS Data Units(EDUs). Each EDU contains a reconstructed CCSDS telemetry packet, quality information, and time stamp. The packetized message transports either real-time or simulated spacecraft telemetry. The Telemetry Subsystem receives spacecraft and instrument Housekeeping (H/K), Health and Safety (H/S), and Diagnostic memory dump data.

FDF: The Flight Dynamics Facility receives real-time, decommutated attitude telemetry. This telemetry subset provides the FDF with spacecraft attitude information which allows the FDF to track and recommend spacecraft orbit adjustments.

FOS Command Subsystem: The Command Subsystem receives decommutated or derived telemetry values that facilitate the verification of real-time and spacecraft stored commands.

FOS User Interface: The User Interface receives and displays decommutated or derived telemetry values, event information, and status information generated by the Telemetry Subsystem.

FOS Analysis Subsystem: The Analysis Subsystem receives decommutated or derived historical telemetry values that facilitate analysis and trending of spacecraft subsystem health and anomalies.

FOS Resource Management Subsystem: The Resource Management Subsystem supplies configuration information required by the Telemetry Subsystem for real-time or historical telemetry processing. This data includes telemetry database selections, EDOS and client communication channels, and user configuration requests. Checkpoint information and user directive status is forwarded from the Telemetry Subsystem to the Resource Management Subsystem.

FOS Data Management Subsystem: As part of the Telemetry Subsystem initialization phase, telemetry database information concerning telemetry decommutation, conversion,

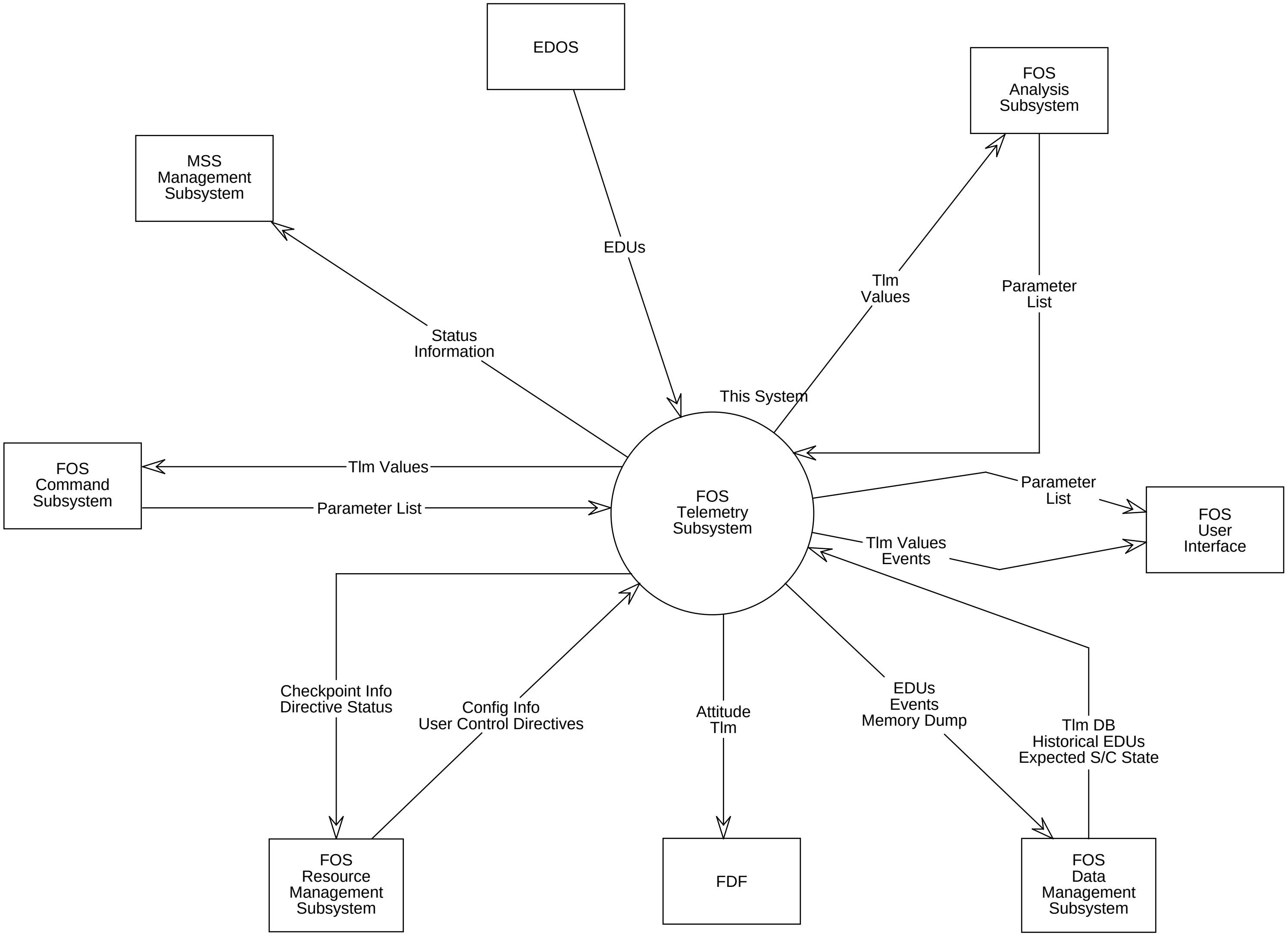

Figure 3.1-1. Telemetry Context Diagram

and checking is retrieved from the Data Management Subsystem. During a real-time spacecraft contact, telemetry EDUs, memory dump data, and telemetry events are forwarded to the Data Management Subsystem for storage and processing. The Data Management Subsystem supplies historical telemetry EDUs to the Telemetry Subsystem during FOS replays or telemetry analysis.

CSMS Management Subsystem: The CSMS Management Subsystem processes messages and collects information pertaining to the status of the Telemetry Subsystem.

3.2 Telemetry Decommutation

The Telemetry Decommutation component provides the capability to decommutate, convert, and limit check housekeeping or engineering telemetry data from the EOS spacecraft and instruments. Telemetry decommutation also provides the mechanisms to calculate derived parameters.

3.2.1Telemetry Decommutation Context

The telemetry decommutation context diagram shown in Figure 3.2-1 depicts the data flows between the FOS Telemetry Subsystem and external ground system as well as EOC internal components. Descriptions of the data flows are summarized for each component:

EDOS: The EDOS forwards telemetry to decommutate via EDOS Data Units (EDUs). Each EDU contains a reconstructed CCSDS telemetry packet, quality information, and time stamp. The packetized message transports either real-time or simulated spacecraft telemetry. Telemetry decommutation receives spacecraft and instrument Housekeeping (H/K) and Health and Safety (H/S).

Parameter Server: The Parameter Server receives parameters from Decom.

FOS Data Management Subsystem: As part of the telemetry decommutation initialization phase, telemetry database information concerning decommutation, conversion, and checking is retrieved from the Data Management Subsystem. During a real-time spacecraft contact, telemetry EDUs, and telemetry events are forwarded to the Data Management Subsystem for storage and processing. The Data Management Subsystem supplies historical telemetry EDUs to the Telemetry Subsystem during FOS replays or telemetry analysis.

FOS Resource Management Subsystem: The Resource Management Subsystem supplies configuration information required by Telemetry Decommutation for real-time or historical telemetry processing. This data includes telemetry database selections, EDOS and client communication channels, and user configuration requests. Checkpoint information and user directive status are forwarded from Telemetry Decommutation to the Resource Management Subsystem.

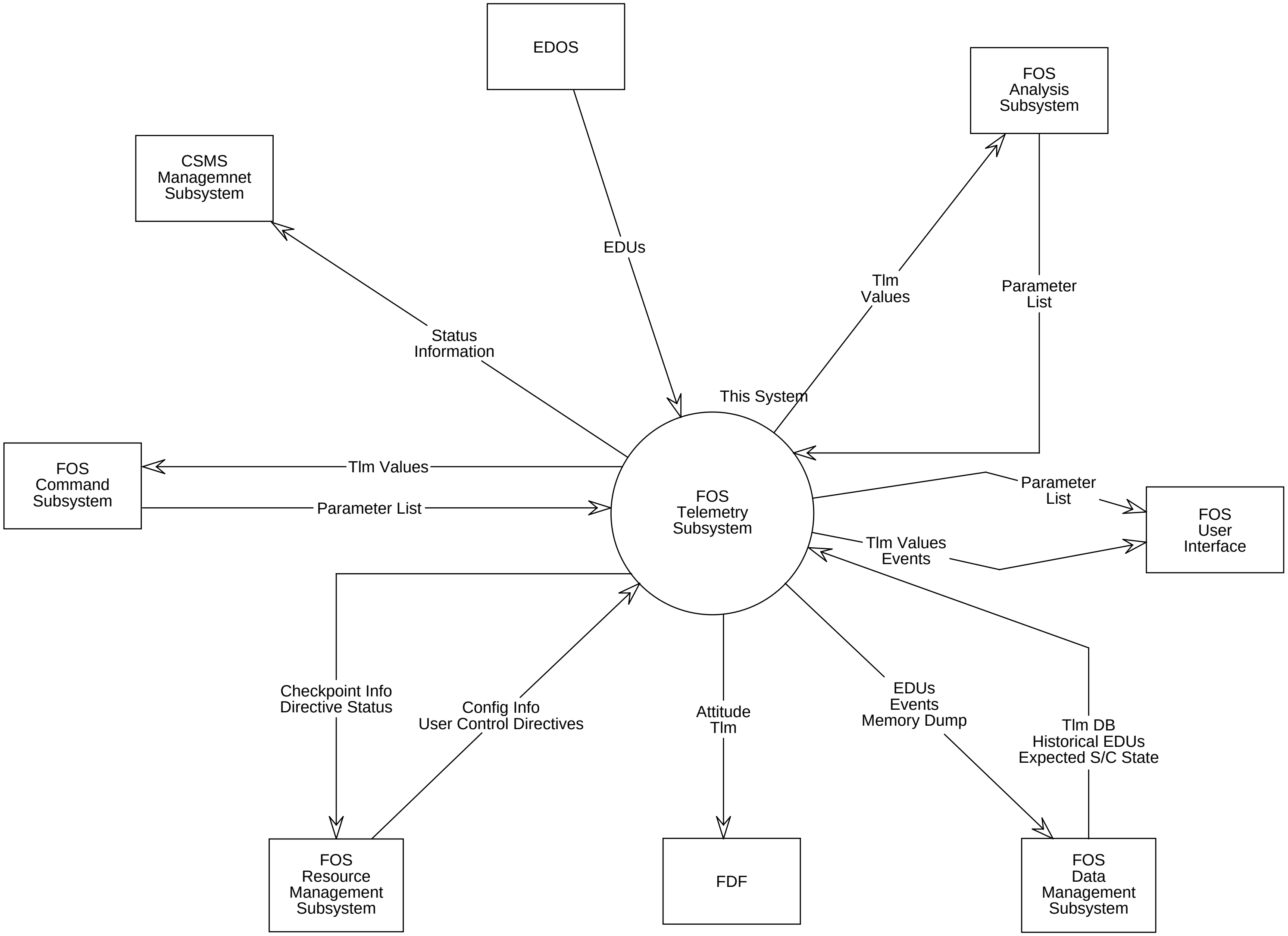

Figure 3.2-1. Telemetry Decommutation Context Diagram

3.2.2 Telemetry Decommutation Interfaces

Table 3.2-1. Telemetry Decommutation Interfaces

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Telemetry Configuration Proxy	FtTlTelemetryConfig	Provides for configuring and adjusting of a telemetry process	TLM	RMS	At initialization of a telemetry process and upon user directive
EDOS interface	FtTlEdu	Provides EDUs for decommutation	TLM	TLM	Every EDU
Parameter Server interface	FoPsClientIF	Provides telemetry parameters to other subsystems	TLM	TLM RMS CMD FUI	As requested by the user
Telemetry Archiver interface	FdArTlmArchProxy	Archives EDUs	DMS	TLM	At every EDU

3.2.3 Telemetry Decommutation Object Model

The telemetry decommutation object model is shown in Figure 3.2-2.

FtTlTelemetryConfig class handles configuration requests from the user. This class is a proxy for the Resource Management System.

FtTlConfigRequest class is the link class used to carry the information from the FtTlDumpConfig proxy to the memory dump process.

FtTlTelemetryController class is the controller of the telemetry decommutation process. This class configures the process and initiates the decommutation process.

FoGnTlmSourceIF class is the telemetry source interface. This class receives the data and handles the communications layer interface.

FtTlEdu class obtains and verifies the critical information from the EDU. If archiving is enabled, this class sends the EDUs to be archived by DMS.

FtTlDecom inherits from the FtTlEdu class which inherits from the FoGnTlmSourceIF class. This class iterates over its FtTlPacketMaps searching for the correct map to use for decommutation.

FtTlPacketMap class contains the maps for the parameters to decommutate. This class iterates over its FtTlParamMaps searching for the correct map to use to decommutate the parameter.

FtTlContextDepMap class contains the maps of all of the possible context dependent maps for a single parameter. This class iterates over its FtTlDecomContextSwitch class searching for the parameter that is the context dependent switch.

FtTlContextSwitch class compares the high and low values of the context switch parameters to determine which one is the correct switch for the context dependent parameter and checks the quality of the context switch parameter.

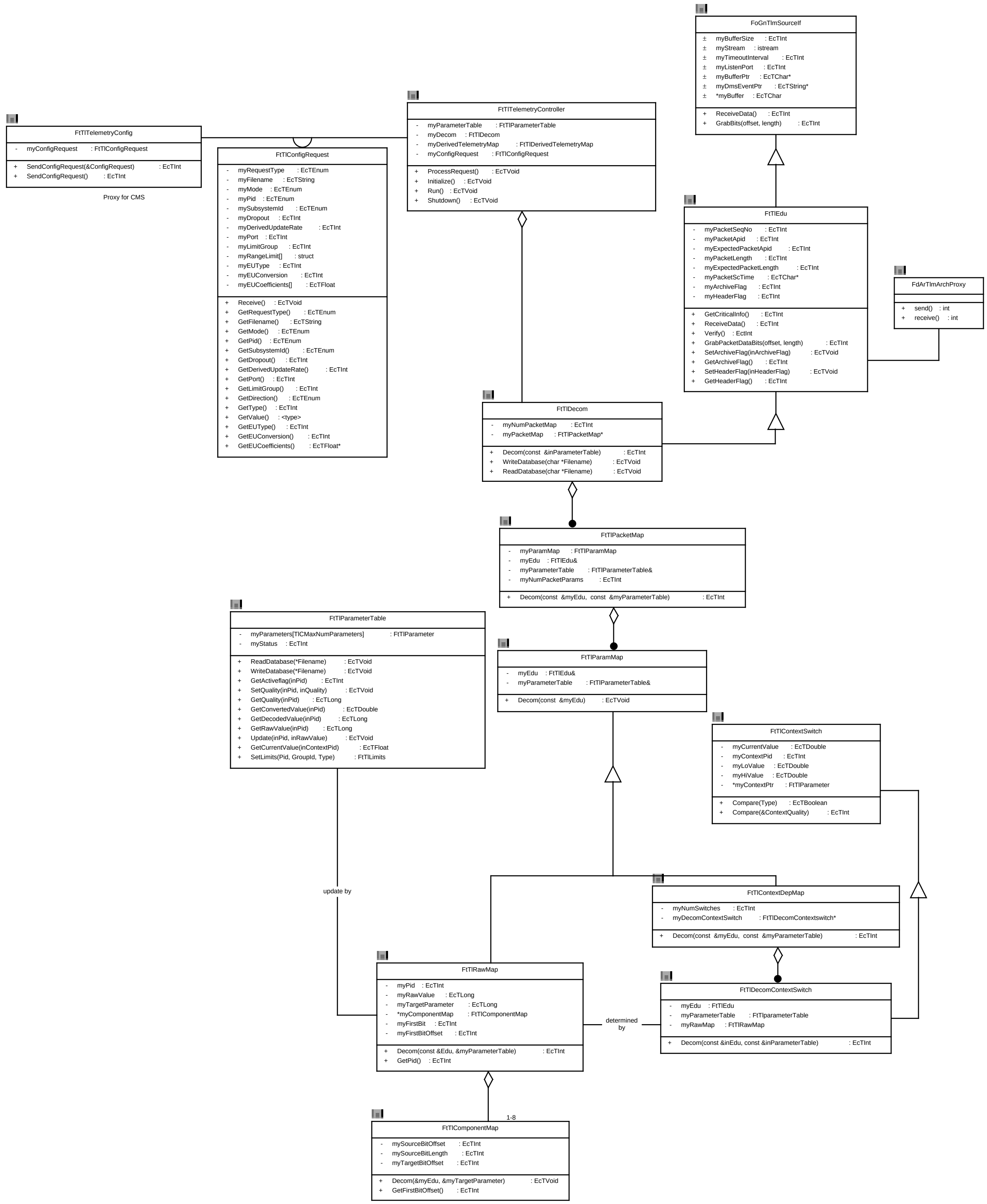

Figure 3.2-2. Telemetry Decommutation Object Model

FtTlDecomContextSwitch class inherits from FtTlContextSwitch. Once the switch is found, the FtTlRawMap can be determined.

FtTlRawMap class iterates over its FtTlComponentMaps searching for the correct place in the target parameter for each component.

FtTlComponentMap class uses the bit offset and the bit length to determine the correct placement of each bit that makes up the parameter. Once the target parameter is filled FtTlParameterTable is updated.

The parameter table object model is shown in Figure 3.2-3.

FtTlParameterTable class is a table that holds all of the telemetry parameters. It has the ability to update different values of the parameter.

FtTlParameterValues class contains all of the values of each parameter that is sent to the parameter server. This class has the ability to retreive each of the values.

FtTlStatus class represents all of the statuses of a parameter. This class is sent along with the FtTlParameterValues to the parameter server.

FtTlParameter class represents the different kinds of parameters. A parameter can be analog or discrete. This class has the ability to retreive the values of the range limits. It can also set the range limit values and the delta limit values. Selective decommutation for a single parameter is set in this class.

FtTlDecode class determines the decoded value of the parameter.

FtTlDeltaLimit class has the delta limit value and checks if the parameter's value has exceeded the delta limit. The delta limit value can also be set in this class.

FtTlDiscreteParam class represents a discrete parameter. Discrete parameters can be range limit checked and may be context dependent.

FtTlAnalogParam class represents a analog parameter. Analog parameters can be EU converted, range limit checked, and may be context dependent. The EU conversion type and the range limit set can be selected by the user.

FtTlConversionSet class represents the conversion sets defined for a parameter. There can be up to four EU conversions defined for each parameter. The current EU conversion can be

selected by the user. FtTlConversionSet iterates over its FtTlEuConversions searching for the algorithm that is selected. If no algorithm is selected, the FtTlParameterContextSwitch is consulted to determine the correct algorithm to use.

FtTlParameterContextSwitch class inherits from FtTlContextSwitch. The FtTlParameterContextSwitch is used to determine the FtTlLimitSet or the FtTlConversionSet to use if one is not already defined or selected by the user.

FtTlPolyConversion class is the polynomial conversion class. This class uses the polynomial equation to EU convert the parameter's decoded value.

FtTlExponentialConversion class is the exponential conversion class. This class uses the exponential conversion equation to EU convert the parameter's decoded value.

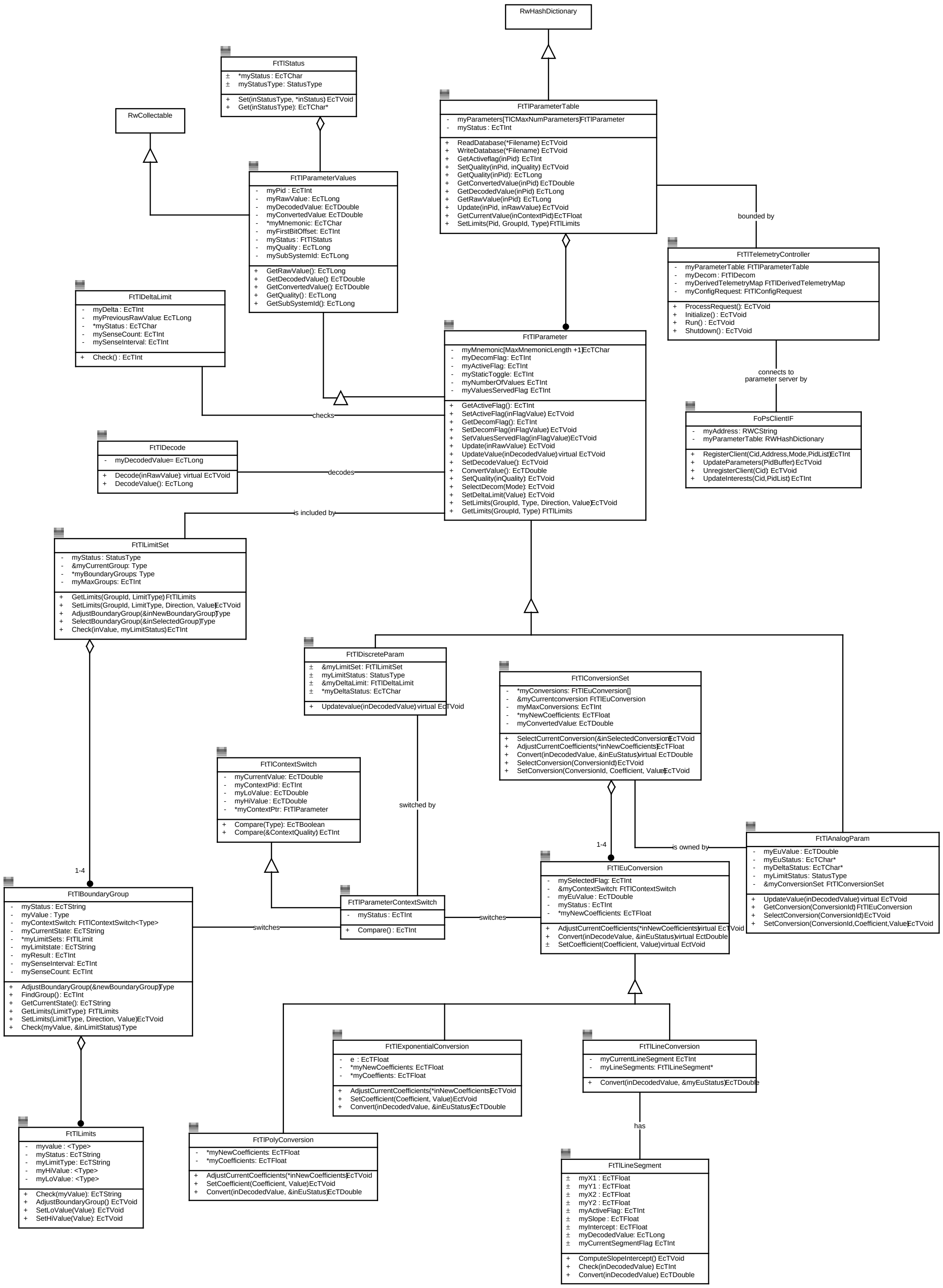

Figure 3.2-3. Parameter Table Object Model

FtTlLineConversion class is the line conversion class. This class iterates over its FtTlLineSegments searching for the correct segment to use to determine the line conversion to use to EU convert the decoded value.

FtTlLineSegment class represents one line segment that can be used to EU convert the decoded value.

FtTlLimitSet class represents the limit sets that are defined for a parameter's range limit checking. If a particular FtTlLimitSet is not defined for the parameter, the parameter's FtTlParameterContextSwitch is consulted to determine the correct FtTlLimitSet to use. FtTlLimitSet contains up to four FtTlBoundaryGroups that can be defined for each parameter.

FtTlBoundaryGroup class represents a boundary group. FtTlBoundaryGroup iterates over its FtTlLimits checking the range limits.

FtTlLimits class contains the high and low range limit values that are used to determine if a parameter's value has violated the range limits. This class also allows the user to set the range limit values.

The Derived Telemetry object model is shown in Figure 3.2-4.

FtTlDerivedTelemetryMap class contains FtTlEquations that are used to calculate derived telemetry parameter values.

FtTlEquation class contains FtTlElements that represents the equation that is used to derive a parameter.

FtTlElement class is an abstract base class that represents all of the possible parts of an equation.

FtTlParamOperand class is the operand of the equation for the derived parameter.

FtTlConstant class represents a constant in an equation for the derived parameter.

FtTlOperator class represents the operator for the equation of the derived parameter.

FtTlAdd class represents the arithmetic addition operator.

FtTlArcCos class represents the arithmetic arc cosine function.

FtTlNegate class the arithmetic unary minus operator.

FtTlArcSin class represents the arithmetic arcsine function.

FtTlGreaterOrEqual class represents the logical greater than or equal to operator.

FtTlSubtract class represents the arithmetic subtraction operator.

FtTlMultiply class represents the arithmetic multiplication operator.

FtTlDivide class represents the arithmetic division operator.

FtTlEqual class represents the logical equality operator.

FtTlNotEqual class represents the logical inequality operator.

FtTlGreater class represents the logical greater than operator.

FtTlTan class represents the arithmetic tangent function.

FtTlArcTan class represents the arithmetic arctangent function.

FtTlCos class represents the arithmetic cosine function.

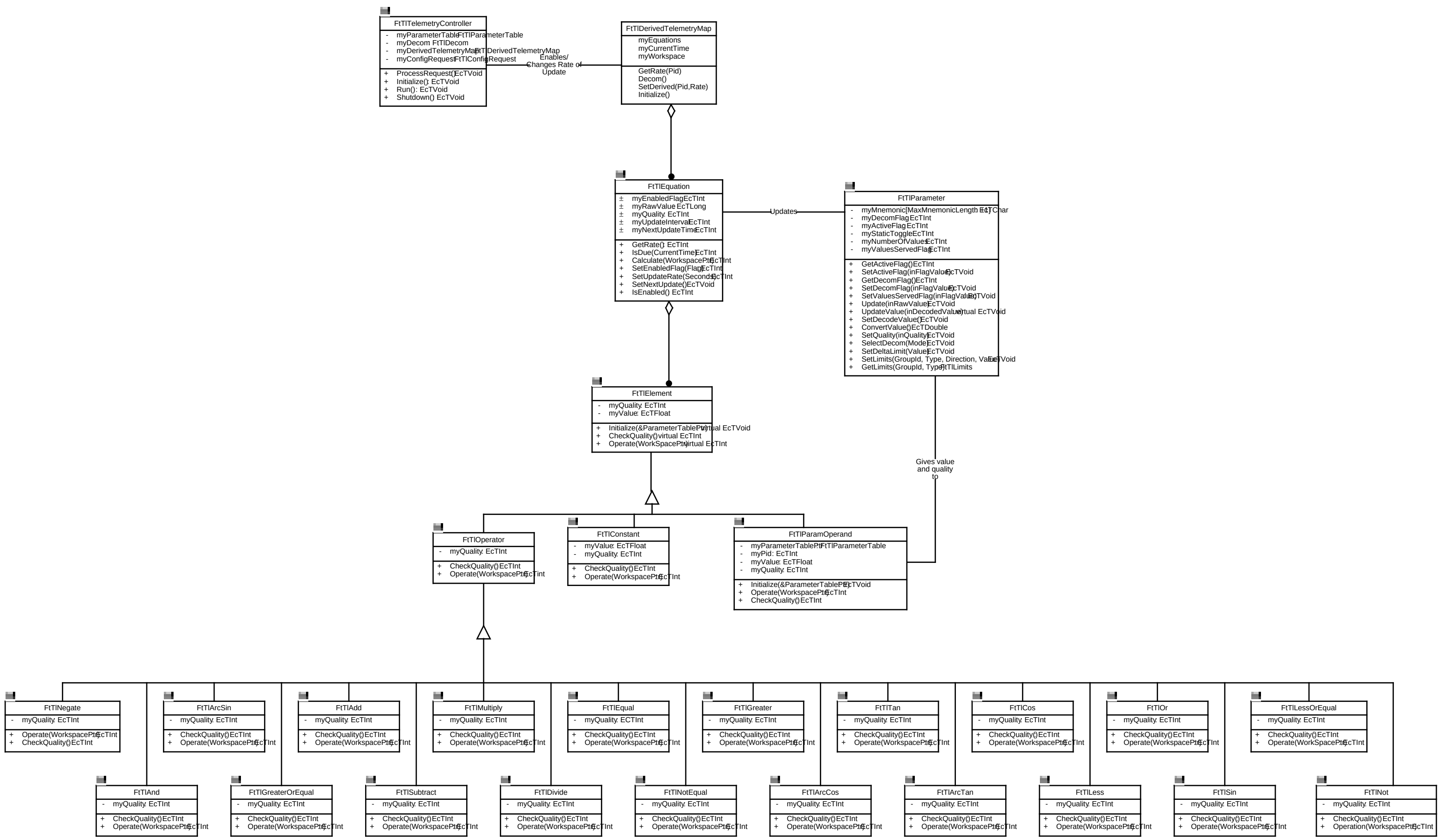

Figure 3.2-4. Derived Telemetry Object Model

FtTlLess class represents the less than operator.

FtTlOr class represents the logical OR operator.

FtTlSin class represents the arithmetic sine function.

FtTlLessOrEqual class represents the logical less than or equal to operator.

FtTlNot class represents the logical NOT operator

3.2.4 Telemetry Decommutation Dynamic Model

The telemetry subsystem is dynamically modeled through scenarios and event trace diagrams depicting the sequence of events to process spacecraft and instrument telemetry. Scenarios for the telemetry subsystem model nominal sequences of events to ingest telemetry data and decommutate parameters from the telemetry. The following scenarios are described in this section:

Decommutate an EDU

Telemetry Derived Parameters

Request to Adjust Limits

Telemetry Dropout

Parameter Updating

Parameter Server Processing

3.2.4.1 Decommutate an EDU Scenario

3.2.4.1.1 Decommutate an EDU Scenario Abstract

The purpose of the decommutate an EDU scenario is to describe the process of building and updating parameters from decommuted telemetry EDUs that are received from the EDOS interface. The event trace for this scenario can be found in Figure 3.2-5.

3.2.4.1.2 Decommutate an EDU Summary Information

Interfaces:

Edos Interface

Stimulus:

FtTlEdu receives EDU data from the EDOS interface.

Desired Response:

Parameters are built and updated from decommutated telemetry.

Pre-Conditions:

The Telemetry Subsystem is configured to accept telemetry EDUs of a particular format from EDOS.

Post-Conditions:

The Telemetry Subsystem is ready to receive additional EDUs.

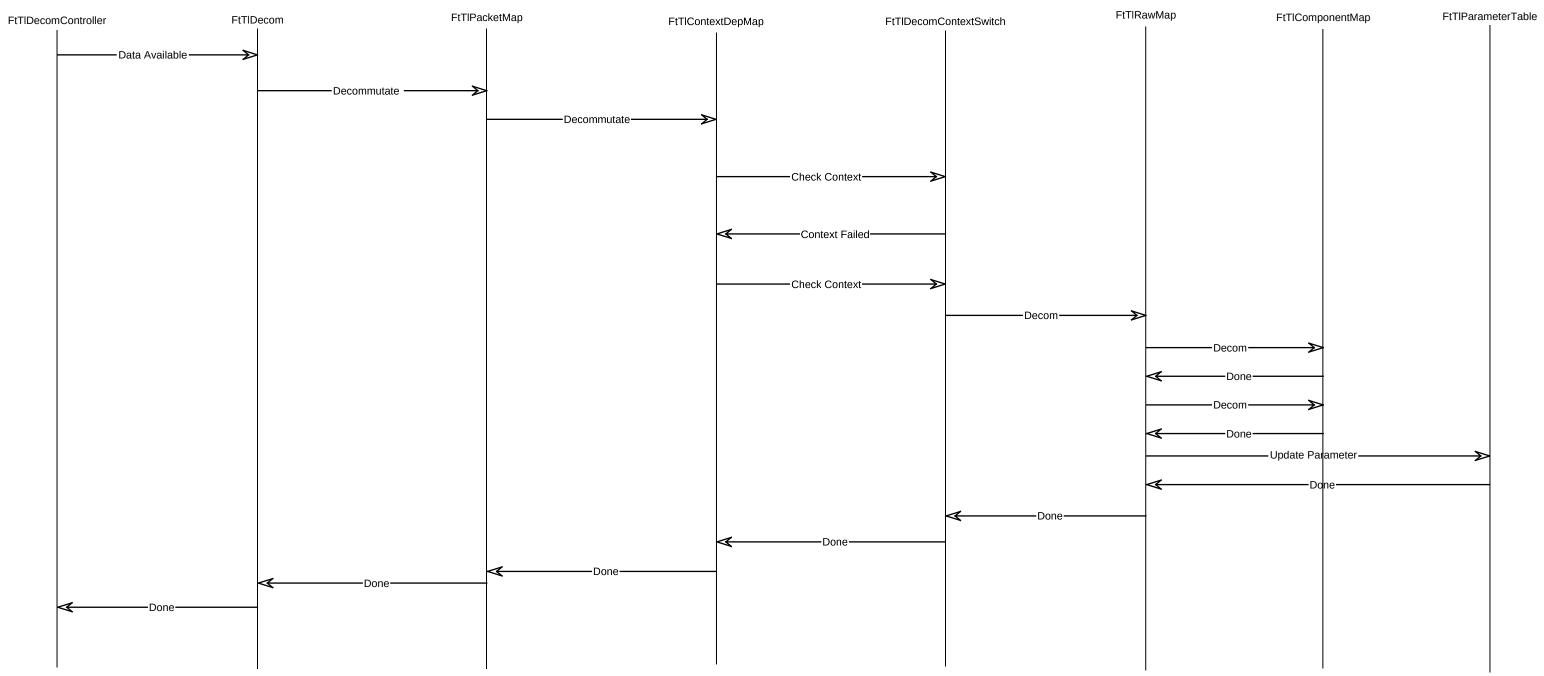

Figure 3.2-5. Decommutate an EDU Event Trace

3.2.4.1.3 Decommutate an EDU Scenario Description

FtTlDecomController initiates the storage and decommutation of a telemetry EDU. FtTlEdu extracts the EDU header fields.The packet APID, length, and sequence count are verified. Next, FtTlDecom initiates the extraction and decommutation of the packet data fields using FtTlPacketMap. FtTlDecom uses information from FtTlEdu to extract the needed bits from the telemetry stream. When a component is context dependent, FtTlContextDepMap initiates the operation which performs the association between the context switched component position and a particular component. If it is determined that context is dependent upon the value of an associated discrete, the switch value is compared to the value of the associated discrete within FtTlContextSwitch until a match is found. When a match is found, the proper FtTlRawMap is called upon to extract the needed bits from the telemetry stream. FtTlRawMap is consulted again for the next component position for the next component to be extracted. FtTlRawMap is asked to assemble a parameter's raw value when all the telemetry information gathering for a given parameter is complete. FtTlParameterTable is called to update the parameter. These events are repeated for each component needed to build each parameter.

3.2.4.2 Select Subsystem Decommutation Mode Scenario

3.2.4.2.1 Select Subsystem Decommutation Mode Scenario Abstract

This scenario describes how to select the mode in a decommutation process. The event trace for this scenario can be found in Figure 3.2-6.

3.2.4.2.2 Select Subsystem Decommutation Mode Summary Information

Interfaces:

RMS

Stimulus:

This scenario occurs when an active decom process is started, or by user directive.

Desired Response:

All parameters within a Subsystem will be decommutated.

Pre-Conditions:

None.

Post-Conditions:

None.

3.2.4.2.3 Select Subsystem Decommutation Mode Scenario Description

FtTlTelemetryConfig sends a FtTlConfigRequest to FtTlTelemetryController requesting that decommutation mode be set within a specified subsystem. FtTlTelemetryController calls FtTlConfigRequest to receive the request. FtTlTelemetryController then calls FtTlConfigRequest to get the request type. When the type is to set the telemetry decommutation mode, FtTlTelemetryController calls FtTlConfigRequest to get the decommutation mode. FtTlTelemetryController then calls FtTlParameterTable to get the FtTlParameter. FtTlTelemetryController then calls FtTlParameter to

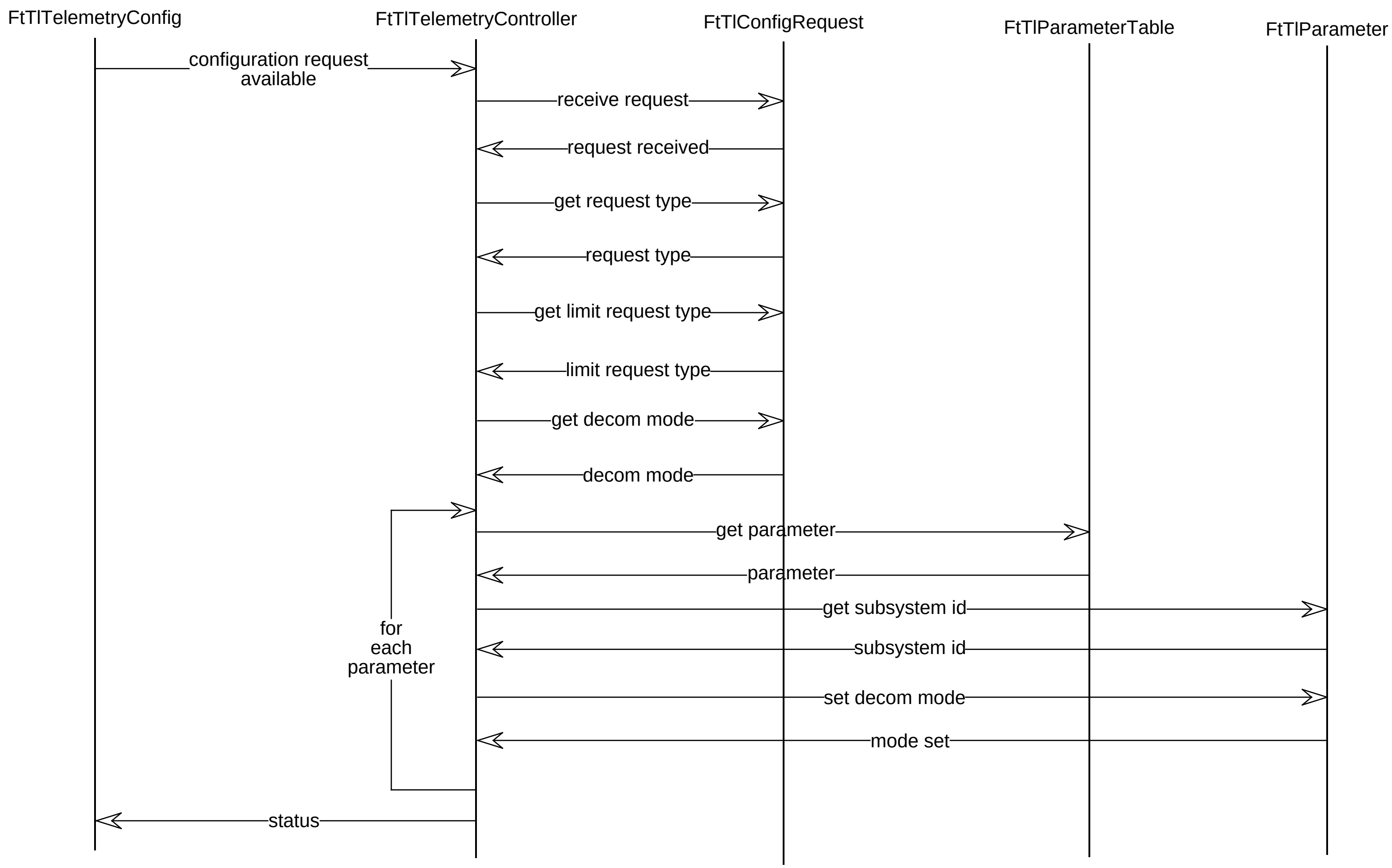

Figure 3.2-6. Select Subsystem Decommutation Mode Event Trace

get the subsystem id of that parameter. If the subsystem id is the same as the requested subsystem id, then FtTlTelemetryController calls FtTlParameter to set the telemetry decommutation mode. When FtTlParameter is finished, FtTlTelemetryController returns to an idle state.

3.2.4.3 Turn Archiving Mode On Scenario

3.2.4.3.1 Turn Archiving Mode On Scenario Abstract

This scenario describes how archiving mode is turned on in a decommutation process. The event trace for this scenario is shown in Figure 3.2-7.

3.2.4.3.2 Turn Archiving Mode On Scenario Summary Information

Interfaces:

RMS

Stimulus:

This scenario occurs when an active decommutation process is started, or by user directive

Desired Response:

All telemetry received by the decommutation process is forwarded to the archiving process.

Pre-Conditions:

None

Post-Conditions:

None

3.2.4.3.3 Turn Archiving Mode On Scenario Description

FtTlTelemetryConfig sends a FtTlConfigRequest to FtTlTelemetryController requesting that archiving be turned on. FtTlTelemetryController calls FtTlConfigRequest to receive the request. FtTlTelemetryController then calls FtTlConfigRequest to get the request type. When the type is to modify archiving, FtTlTelemetryController calls FtTlConfigRequest to get the archiving mode. FtTlTelemetryController then calls FtTlDecom to set its archiving mode to the indicated mode. When this mode is ON, the archiving is turned on. When FtTlDecom is finished, FtTlTelemetryController returns to an idle state.

3.2.4.4 Read a Database Scenario

3.2.4.4.1 Read a Database Scenario Abstract

This scenario describes how a decommutation process reads in a database. The event trace for this scenario is shown in Figure 3.2-8.

3.2.4.4.2 Read a Database Scenario Summary Information

Interfaces:

RMS

Stimulus:

This scenario occurs when the decommutation process is started, or by user directive

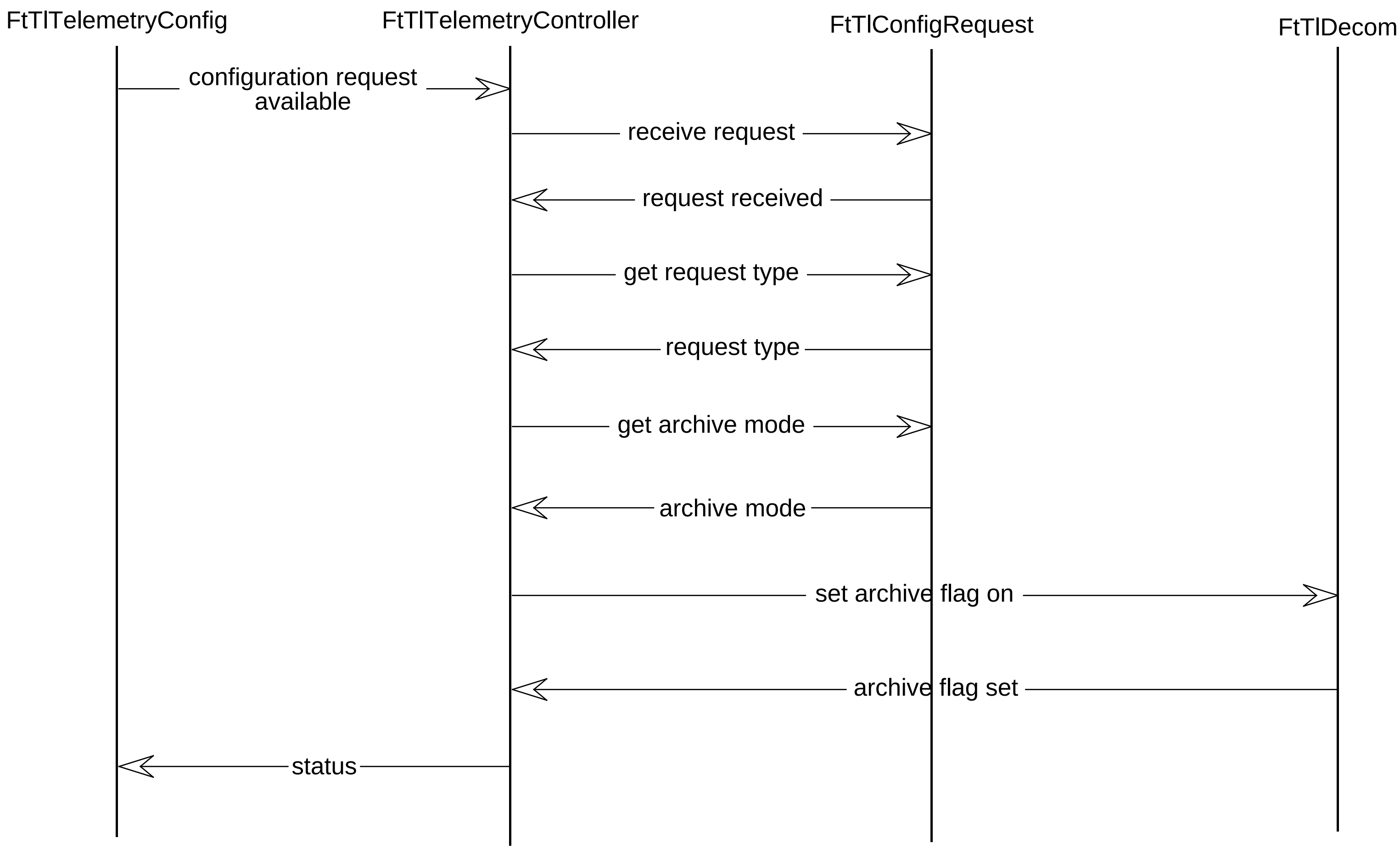

Figure 3.2-7. Turn Archiving Mode On Event Trace

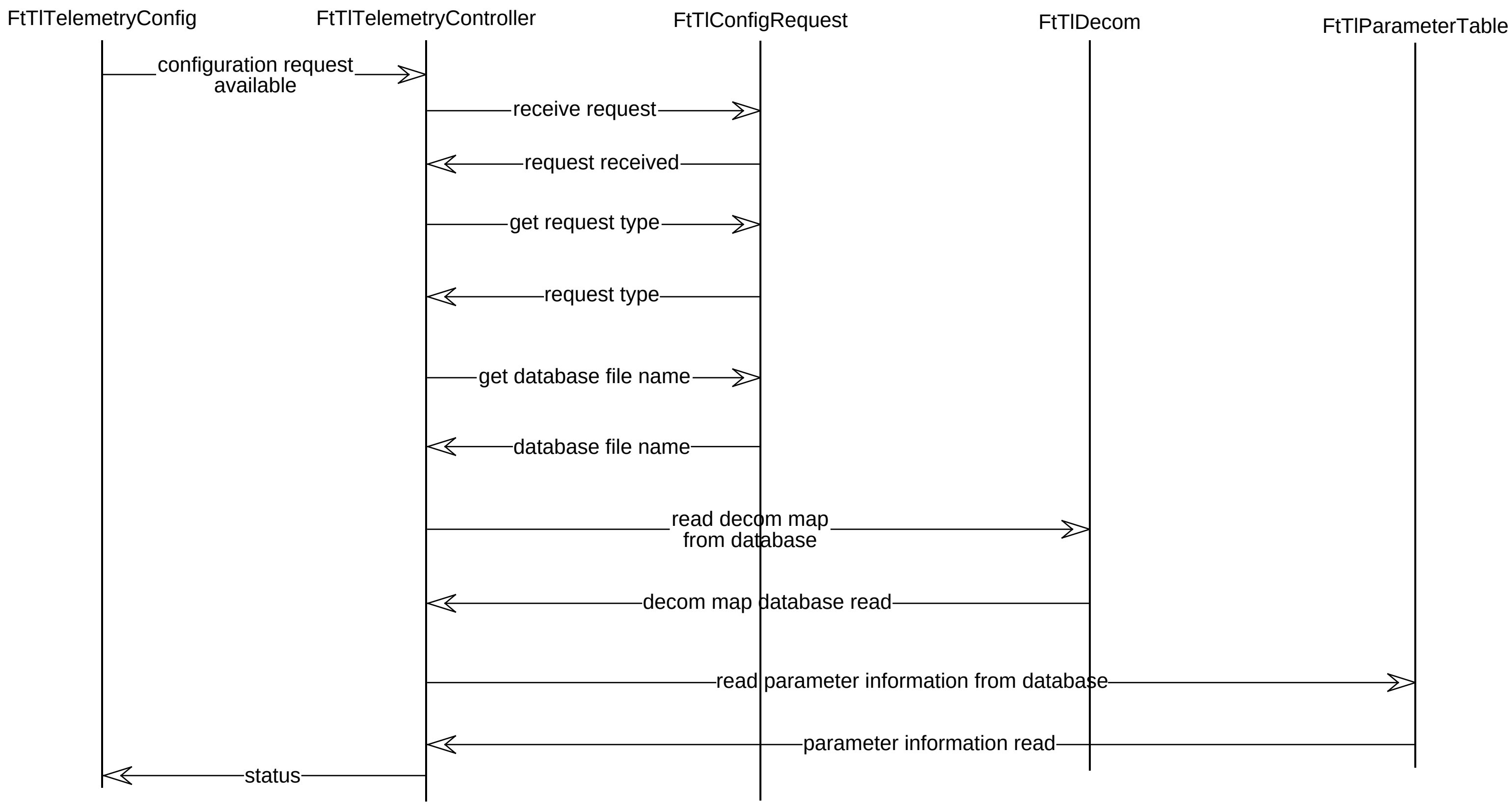

Figure 3.2-8. Read a Database Event Trace

Desired Response:

A new database is loaded into the decommutation process.

Pre-Conditions:

Not in the middle of a decommutation session.

Post-Conditions:

None

3.2.4.4.3 Read a Database Scenario Description

FtTlTelemetryConfig sends a FtTlConfigRequest to FtTlTelemetryController requesting a new database to be read in. FtTlTelemetryController calls FtTlConfigRequest to receive the request. FtTlTelemetryController then calls FtTlConfigRequest to get the request type. When the type is to read in a new database FtTlTelemetryController calls FtTlConfigRequest to get the database file name. FtTlTelemetryController then calls FtTlDecom, telling it to read in the decommutation maps from the indicated database. When FtTlDecom is finished reading, FtTlTelemetryController calls FtTlParameterTable to read in the parameter information from the database. When FtTlParameterTable is finished FtTlTelemetryController returns to an idle state.

3.2.4.5 Telemetry Derived Parameters Scenario

3.2.4.5.1 Telemetry Derived Parameters Scenario Abstract

The purpose of the telemetry derived parameters scenario is to describe the process by which a parameter is updated with a calculated derived telemetry value. The event trace for this scenario can be found in Figure 3.2-9.

3.2.4.5.2 Telemetry Derived Parameters Summary Information

Interfaces:

No External Interfaces

Stimulus:

An EDU has been fully decommutated.

Desired Response:

FtTlParameterTable will contain a calculated derived telemetry value.

Pre-Conditions:

All EDU telemetry samples have been decommutated and the packet's spacecraft time stamp is available.

Post-Conditions:

Updated derived parameter values are put into the parameter table.

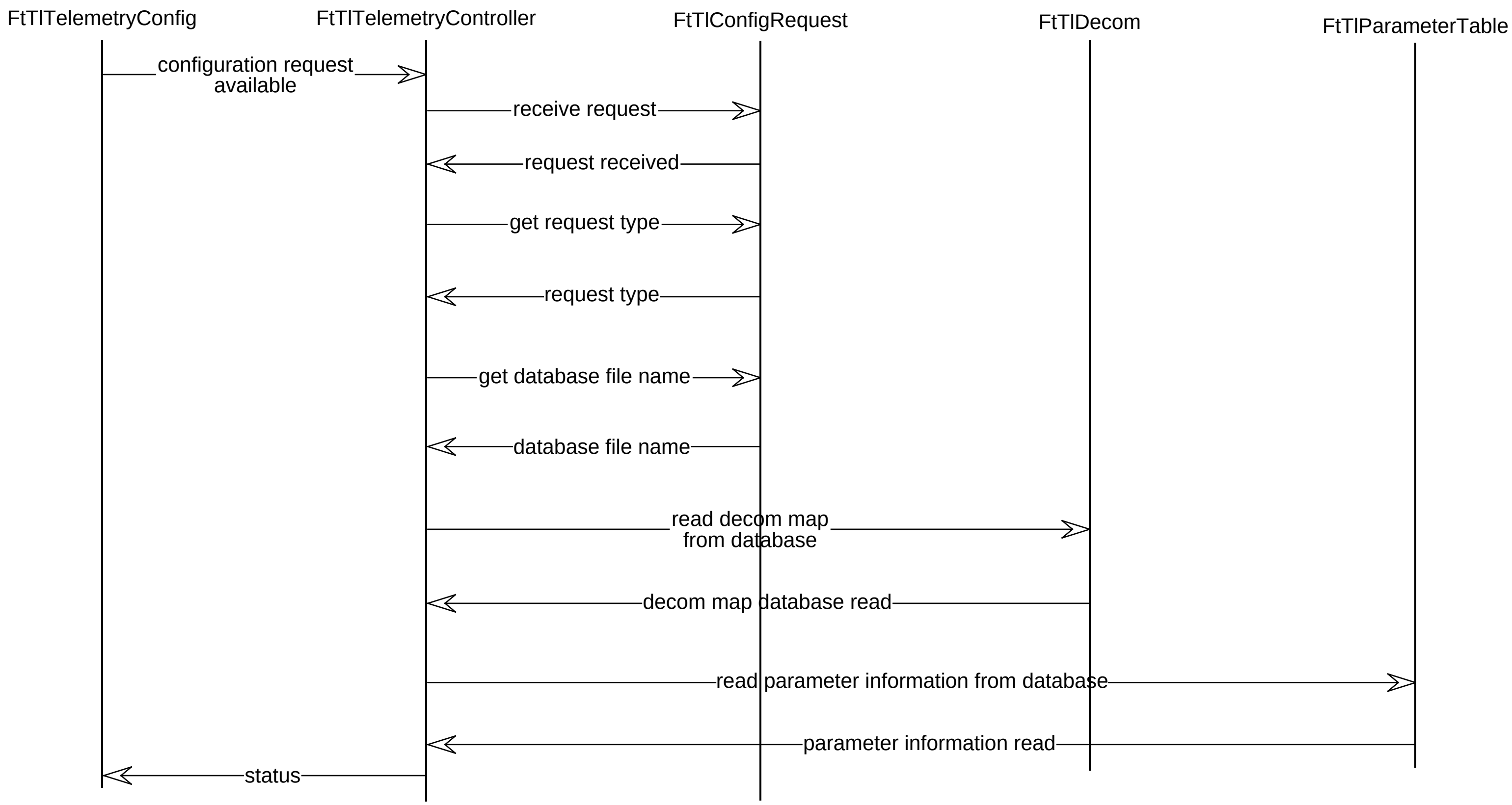

Figure 3.2-9. Telemetry Derived Parameters Event Trace

3.2.4.5.3 Telemetry Derived Parameters Scenario Description

FtTlDecomController initiates processing of the derived telemetry parameters that are constructed using downlink telemetry and predefined constant information. FtTlDerivedTelemetryMap checks if a FtTlEquation is enabled. If the FtTlEquation is enabled, FtTlDerivedTelemetryMap checks if the FtTlEquation is due for update. By comparing the current time with the equation's next update time, FtTlEquation determines whether the equation is to be calculated and sends a response to FtTlDerivedTelemetryMap. When FtTlDerivedTelemetryMap receives confirmation, FtTlEquation verifies if each FtTlParamOperand is of good quality. If the FtTlParamOperand is good, FtTlEquation uses the operate member function of each FtTlElement to calculate the equations and update FtTlParameter. FtTlParameter contains the calculated derived telemetry value and indicates when the calculation is complete. These events are repeated for subsequent equations.

3.2.4.6 Set Polynomial Coefficients for EU Conversion Scenario

3.2.4.6.1 Set Polynomial Coefficients for EU Conversion Scenario Abstract

This scenario describes how to set the engineering unit conversion algorithm coefficients in a decommutation process. The event trace for this scenario is shown in Figure 3.2-10.

3.2.4.6.2 Set Polynomial Coefficients for EU Conversion Summary Information

Interfaces:

Resource Management.

Stimulus:

By user directive.

Desired Response:

The coefficients in the indicated engineering unit conversion algorithm will be replaced with a new set of coefficients.

Pre-Conditions:

None

Post-Conditions:

None

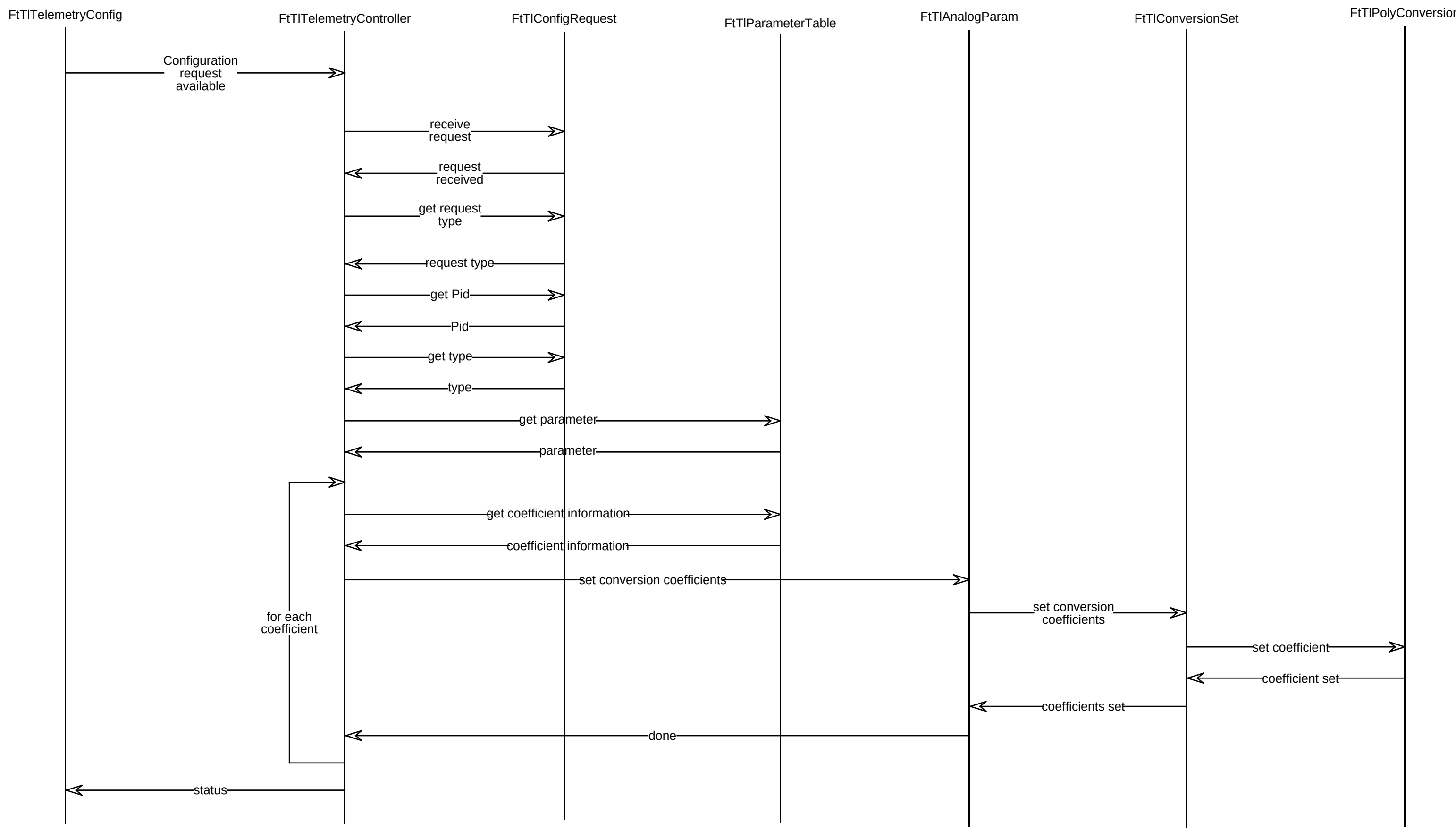

Figure 3.2-10. Set Polynomial Coefficients for EU Conversion Event Trace

3.2.4.6.3 Set Polynomial Coefficients for EU Conversion Scenario Description

FtTlTelemetryConfig sends a FtTlConfigRequest to FtTlTelemetryController requesting that the engineering unit conversion algorithm coefficients be set for a specified algorithm. FtTlTelemetryController calls FtTlConfigRequest to receive the request. FtTlTelemetryController then calls FtTlConfigRequest to get the request type. When the type is to change the EU conversion coefficients, FtTlTelemetryController calls FtTlConfigRequest to get the Pid of the parameter whose coefficient we want to change. FtTlTelemetryController then calls FtTlParameterTable to get the FtTlAnalogParam. For each coefficient that is to be changed, FtTlTelemetryController calls FtTlConfigRequest to get the ConversionId, the CoefficientId and the CoefficientValue, then calls FtTlAnalogParam to set the conversion coefficients of that parameter. FtTlAnalogParam calls FtTlConversionSet to set the conversion coefficients. FtTlConversionSet uses the ConversionId to determine which EU Conversion algorithm to modify and calls FtTlPolyConversion to set the coefficient specified in CoefficientId to the value specified in CoefficientValue. When FtTlTelemetryController completes looping through each coefficient to be changed, it returns to an idle state.

3.2.4.7 Request to Adjust Limits Scenario

3.2.4.7.1 Request to Adjust Limits Scenario Abstract

The purpose of the Request to Adjust Limits Scenario is to describe the process by which a request to update range limits is handled. The event trace for this scenario can be found in Figure 3.2-11.

3.2.4.7.2 Request to Adjust Limits Summary Information

Interfaces:

Resource Management.

Stimulus:

By user directive.

Desired Response:

The range limits indicated will be replaced with a new set of coefficients.

Pre-Conditions:

None

Post-Conditions:

None

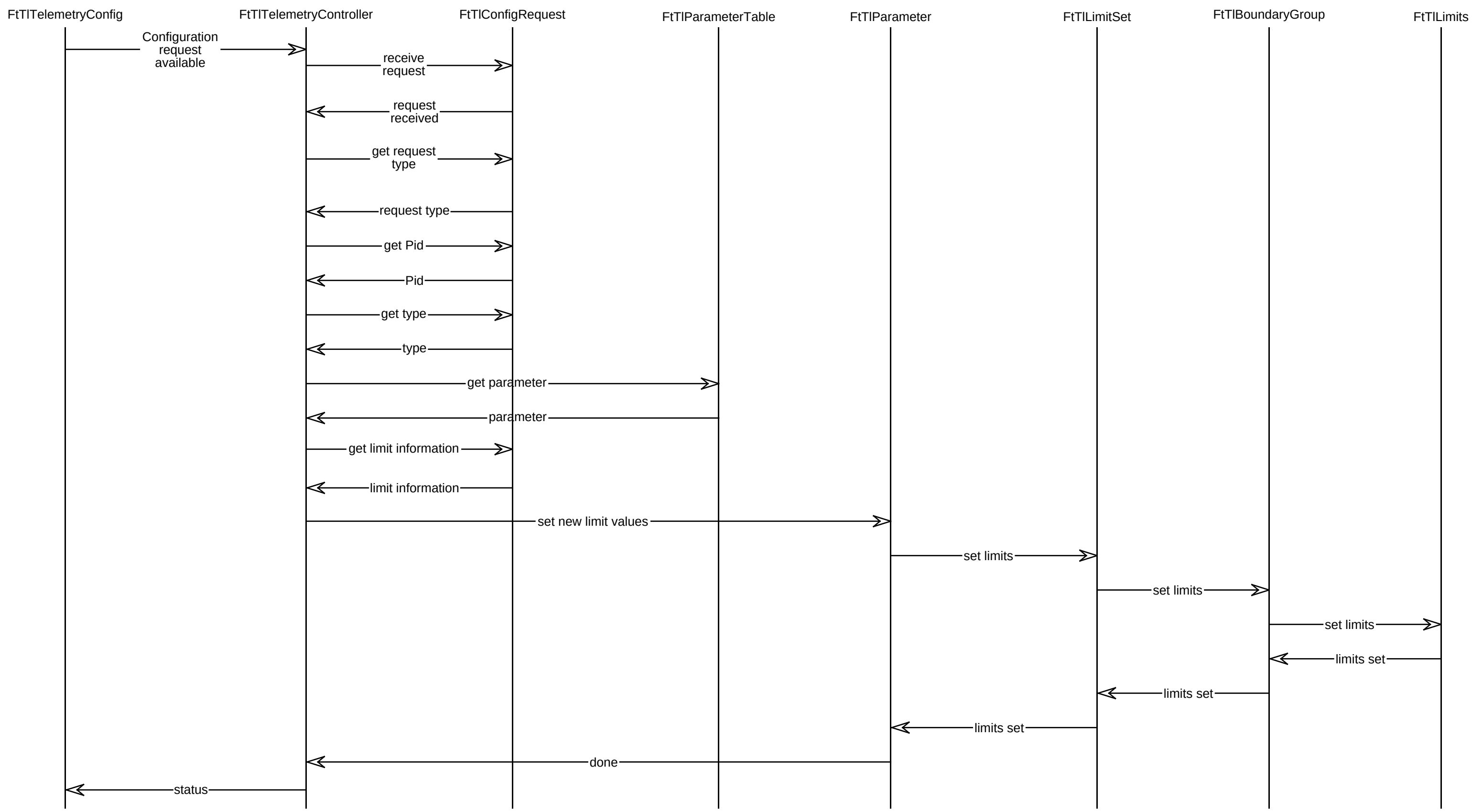

Figure 3.2-11.   Request to Adjust Limits Event Trace

3.2.4.7.3 Request to Adjust Limits Scenario Description

FtTlTelemetryConfig sends a FtTlConfigRequest to FtTlTelemetryController requesting that the range limits be adjusted for a specified parameter. FtTlTelemetryController calls FtTlConfigRequest to receive the request. FtTlTelemetryController then calls FtTlConfigRequest to get the request type. When the type is to adjust parameter range limits, FtTlTelemetryController calls FtTlConfigRequest to get the Pid of the parameter whose limits we want to change, and also to get the type of limit request this is. FtTlTelemetryController then calls FtTlParameterTable to get the FtTlParameter indicated by the Pid. When the type of limit request is range limits, then for each range limit that is to be changed, FtTlTelemetryController calls FtTlConfigRequest to get the GroupId, RangeLimitType, LimitDirection and the new LimitValue, then calls FtTlParameter to set the limits using these new values. FtTlParameter calls FtTlLimitSet to set the limits. FtTlLimitSet uses the GroupId to determine which boundary group to use and calls FtTlBoundaryGroup to set the limits. FtTlBoundaryGroup uses the direction to determine if this is a high or low value to be set and it uses the RangeLimitType to determine the type of limit (i.e. Red, Yellow, etc.) and then calls FtTlLimits to set the actual value. When each range limit is changed, FtTlTelemetryController returns to an idle state.

3.2.4.8 Obtain Current Limit Values Scenario

3.2.4.8.1 Obtain Current Limit Values Scenario Abstract

The purpose of this scenario is to describe the process by which a client can obtain the limits of a given parameter. The event trace for this scenario is shown in Figure 3.2-12.

3.2.4.8.2 Obtain Current Limit Values Scenario Summary Information

Interfaces:

Resource Management.

Stimulus:

A client request to obtain the limits of a given parameter.

Desired Response:

The limits of a given parameter are obtained.

Pre-Conditions:

None.

Post-Conditions:

None.

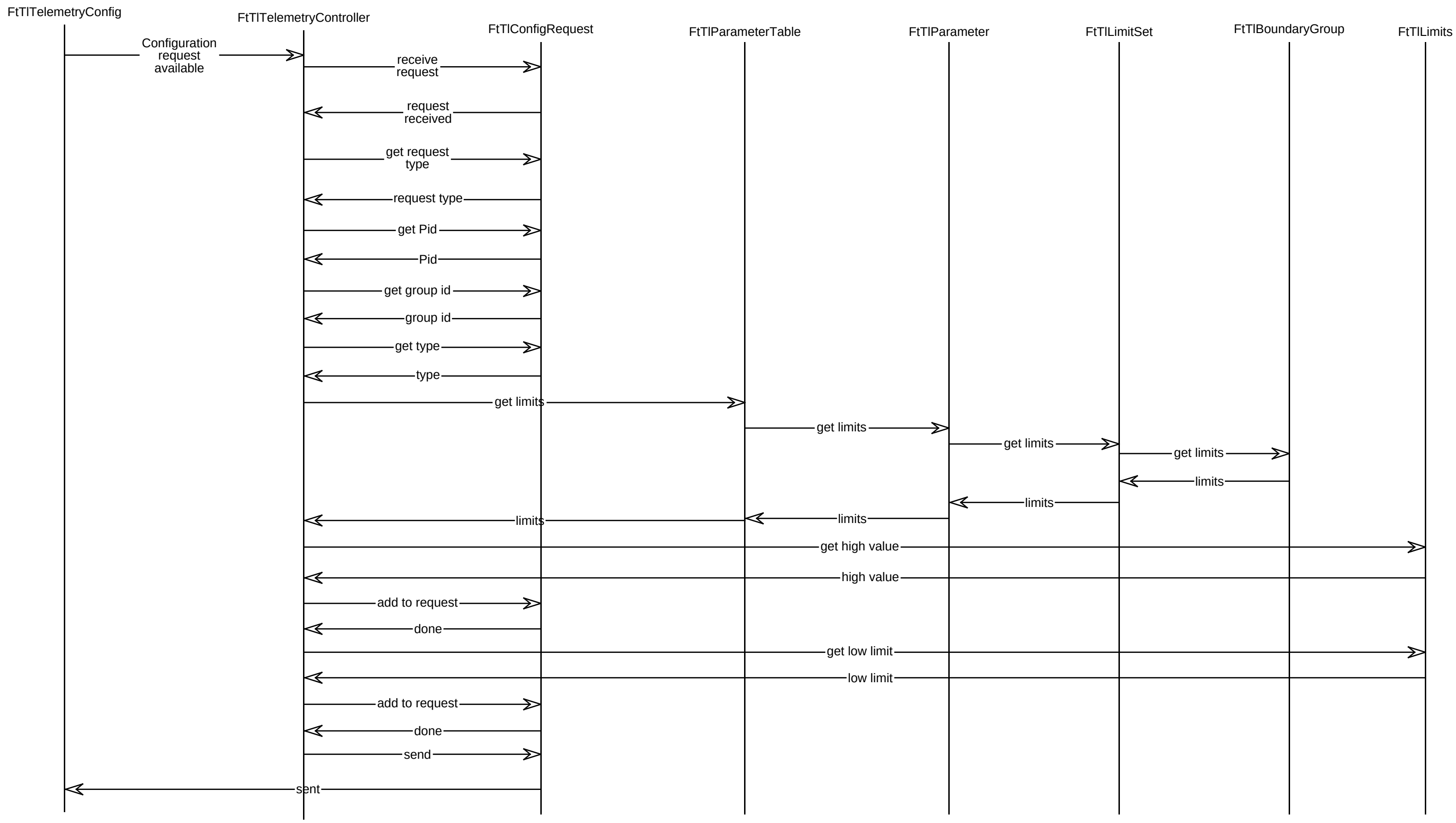

Figure 3.2-12. Obtain Current Limit Values Event Trace

3.2.4.8.3 Obtain Current Limit Values Scenario Description

The client sends a directive to the FtTlTelemetryController through the FtTlTelemetryConfig proxy object by packaging the directive in a FtTlConfigRequest object. The FtTlTelemetryController receives the FtTlConfigRequest object and determines the type. The type is determined to be a limits directive with the objective to obtain the limits of a given parameter for display. The FtTlTelemetryController calls the GetLimits operation of the FtTlParameterTable. The FtTlParameterTable then calls its GetParameter to get the parameter whose limits we want to obtain. The FtTlParameterTable then calls FtTlParameter's GetLimits operation. The FtTlParameter knows of its limit sets and then calls the GetLimits function of FtTlLimitSet. The FtTlLimitSet can then select the appropriate boundary group and call the GetLimits function of the FtTlBoundaryGroup. This returns the correct FtTlLimits object back to the FtTlTelemetryController. The FtTlTelemetryController then calls the GetHiValue and the GetLoValue functions of the FtTlLimits to add the information to the FtTlConfigRequest to be sent back to the client. The decommutation process goes back to blocking on incoming requests.

3.2.4.9 Parameter Updating Scenario

3.2.4.9.1 Parameter Updating Scenario Abstract

The purpose of the Parameter Updating scenario is to describe the process by which a parameter is updated with EU converted and limit checked values. The event trace for this scenario can be found in Figure 3.2-13.

3.2.4.9.2 Parameter Updating Summary Information

Interfaces:

No external interfaces.

Stimulus:

A raw value for an FtTlAnalogParam has been decommutated.

Desired Response:

Parameter updating completes.

Pre-Conditions:

All parameter raw bits have been extracted and assembled.

Post-Conditions:

A parameter is EU converted, limit checked, and ready to be sent to the users. The Telemetry Subsystem is ready to process additional parameters.

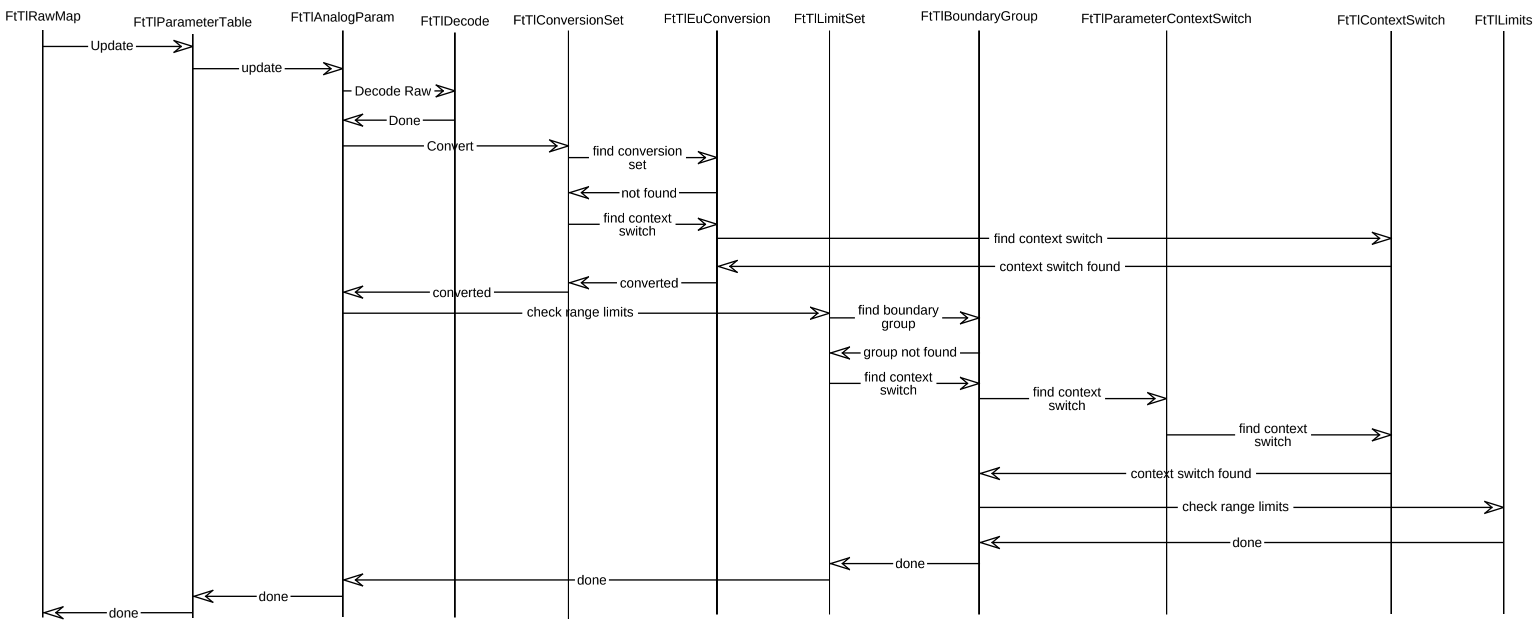

Figure 3.2-13. Parameter Updating Event Trace

3.2.4.9.3 Parameter Updating Scenario Description

FtTlRawMap gives the raw value to FtTlParameterTable and instructs the object to update. The updating involves setting the raw value, performing engineering unit conversion and checking limits. FtTlParameterTable instructs FtTlParameter to decode the raw value using FtTlDecode. FtTlAnalogParam instructs FtTlConversionSet to convert the raw value. FtTlConversionSet uses the FtTlEuConversion objects selected by the user. If none have been selected, each FtTlEuConversion object consults its FtTlParameterContextSwitch. FtTlParameterContextSwitch consults its FtTlContextSwitch to check if that context switch parameter is of bad quality or is marked static. If the context switch is of good quality and is not marked static, FtTlContextSwitch compares its low and high values to an associated FtTlDiscreteParam's value. If the discrete parameter's value falls in the range between the two context switch values, the associated FtTlEuConversion is selected. Otherwise, the next FtTlEuConversion consults its FtTlParameterContextSwitch until a conversion is selected. FtTlEuConversion then converts the raw value to an EU value which is then passed back to the FtTlConversionSet which passes the value to FtTlAnalogParam. Once the conversion has been accomplished, FtTlAnalogParam instructs FtTlLimitSet to begin limit checking. As with the conversion, FtTlLimitSet uses the FtTlBoundaryGroup selected by the user. Not having found one, each FtTlBoundaryGroup consults its FtTlParameterContextSwitch until a boundary group is selected. FtTlBoundaryGroup then calls FtTlLimits to compare either the raw or the EU value with values corresponding to various states: rail low, red low, yellow low, yellow high, red high and rail high. This status is then passed back to FtTlBoundaryGroup. If the status is an alarm or warning state, mySenseCount is incremented. If mySenseCount exceeds mySenseInterval, an event message is generated. The status is then passed back to FtTlLimitSet and then to FtTlAnalogParam. Parameter updating is complete.

3.2.5Telemetry Decommutation Data Dictionary

FdArTlmArchProxy - class that is a proxy from DMS that archives EDUs.

FoGnTlmSourceIF - class the initiates initialization of connections through a port. It receives a data stream, checks for errors and writes the data to a buffer.

> **myBuffer** - attribute that stores the data.
>
> **myBufferPtr** - attribute that points to the location of the data in the data buffer.
>
> **myBufferSize** - attribute that indicates the size of the buffer.
>
> **myDmsEventPtr** - attribute that points to an event message when an error has occurred.
>
> **myListenPort** - attribute that represents the listening port number.
>
> **myStream** - attribute that represents the data stream.
>
> **myTimeoutInterval** - attribute that represents the time interval between data.
>
> **GrabBits** - operation that extracts bits from the buffer.
>
> **ReceiveData** - operation that fills the buffer with data.

FtTlAdd - this class represents the arithmetic addition operator.

> **CheckQuality** - this member function will check the quality of an operator.
>
> **myQuality** - this member variable holds the quality information for an operator.

Operate - this member function will remove two values from a stack and add them together.

FtTlAnalogParam - class that corresponds to the analog parameter type. Analog parameters may be engineering unit converted, boundary limit checked and delta limit checked.

ConversionSet - attribute that points to the conversion set associated with the parameter.

GetConversion - operation that returns a conversion.

myConvertedValue - attribute that contains the converted value.

myDeltaStatus - attribute that is the delta limit check status.

myEuStatus - attribute that holds the current EU conversion status.

myEuUnit - attribute that indicates the units in which the converted value is to be interpreted.

myEuValue - attribute that holds the current EU converted value of the parameter.

myLimitStatus - attribute that points to the limit set associated with the parameter.

SelectConversion - operation that sets the conversion to use.

SetConversion - operation that configures the conversions.

UpdateValues - operation that updates the parameter value from the decoded value.

FtTlAnd - this class represents the logical AND operator.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove two values from a stack and AND them.

myQuality - this member variable holds the quality information for an operator.

FtTlArcCos - this class represents the arithmetic arccosine function.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove a value from a stack and evaluate the arccosine of that value.

myQuality - this member variable holds the quality information for an operator.

FtTlArcSin - this class represents the arithmetic arcsine function.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove a value from a stack and evaluate the arcsine of that value.

myQuality - this member variable holds the quality information for an operator.

FtTlArcTan - this class represents the arithmetic arctangent function.

myQuality - this member variable holds the quality information for an operator.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove a value from a stack and evaluate the arctangent of that value.

FtTlBoundaryGroup - class that represents a limit boundary group. The boundary group contains the high and low warning and alarm boundary values.

myRailLimitsFlag - indicates if rail limits are defined for the parameter.

myContextSwitch - attribute that, if present, provides an association between the boundary group and a context switch. Context switching is based upon the value of a discrete parameter.

myCurrentState - attribute that contains the parameter's current limit state.

myLimitSets - attribute that contains an array of limit types.

myLimitState - attribute that contains the type of limit violation that occurred.

myResult- attribute that indicates if a context switch was found for this parameter.

mySenseInterval- attribute that determines how often an out of limits event message is generated.

mySenseCount - attribute that contains the number of times a parameter has been out of a specific type of limits.

myStatus - attribute that contains the status of the boundary limit check.

myValue - attribute that contains the value that is being limit checked.

Check - operation that performs the boundary limit check.

FindGroup - operation that finds the boundary group using the context dependent parameters.

GetCurrentState - operation that returns the current limit state.

GetLimits - operation that returns a set of limits.

SetLimit - operation that sets the limit of a certain type.

AdjustBoundaryGroup - operation that allows the user to adjust the limit values.

FtTlComponentMap - class that represents the component map. It gets the raw value.

mySourceBitOffset - attribute that contains the bit offset of the component.

mySourceBitLength - attribute that contains the length of the component.

myTargetBitOffset - attribute that contains the offset where the component fits into the parameter.

Decom - operation that builds the parameter with the components.

GetFirstBitOffset - operation that returns the first bit's offset.

FtTlConfigRequest - class that corresponds to configuration update requests.

myDerivedUpdateRate - attribute that contains the rate of updating derived parameters.

myDropout - attribute that contains the dropout interval.

myEUCoefficients - attribute that contains the EU coefficients.

myEUConversion - attribute that contains the EU conversion indicator.

myEUType - attribute that contains the EU conversion type.

myFileName - attribute that contains the filename used for a WriteDatabase or ReadDatabase request.

myLimitGroup - attribute that contains the limit group to set.

myMode - attribute that contains the on or off mode used for archiving or selective decom.

myPid - attribute that contains the parameter identification.

myPort - attribute that contains the input telemetry port.

myRangeLimit - attribute that contains the range limit information.

myRequestType - attribute that contains the type of request.

mySubsystemId - attribute that contains the subsystem identification.

GetDerivedUpdateRate - operation that returns the derived update rate.

GetDirection - operation that returns the range limit direction.

GetDropout - operation that returns the dropout interval.

GetEUCoefficients - operation that returns the EU coefficients.

GetEUConversion - operation that returns the EU conversion.

GetEUType - operation that returns the EU type.

GetFileName - operation that returns the filename.

GetLimitGroup - operation that returns the limit group.

GetMode - operation that returns the mode.

GetPid - operation that returns the parameter identification.

GetPort - operation that returns the telemetry port.

GetRequestType - operation that returns the request type.

GetSubsystemId - operation that returns the subsystem identification.

GetType - operation that returns the range limit type.

GetValue - operation that returns the range limit value.

Receive - operation that receives the data from an external interface.

FtTlConstant - this class represents constant values used in FtTlEquations for derived telemetry.

CheckQuality - this member function will check the quality of a constant.

Operate - this member function will place the constant on a stack for equation processing.

myValue - this member variable holds the value of the constant.

myQuality - this member variable holds the quality information for this constant.

FtTlContextDepMap - class that represents the map used to determine the context switch parameter.

myNumSwitches - attribute that contains the parameter's number of switches.

myDecomContextSwitch - attribute that is an array of decommutation context switches.

mySwitchNotFound - attribute that indicates if a context switch was not found for the parameter.

Decom - operation that initiates the search for the switch mnemonic.

FtTlContextSwitch - class that corresponds to a switch that is used to change the context of an associated class. It can be used to alter the context of a telemetry stream position, an EU conversion, or a limit boundary group.

myContextId - attribute that indicates the parameter ID that is used to enable or disable the context switch

myContextPtr - attribute that is a pointer to the context parameter.

myLoValue - attribute that represents the minimum value used in determining the context switch

myHiValue - attribute that represents the maximum value used in determining the context switch.

myCurrentValue - attribute that represents the current value of the parameter.

Compare - operation that compares its high and low values with the parameter's value to determine the context switch.

FtTlConversionSet - class that represents the set of EU conversions available to an associated analog parameter.

myConversions - attribute that contains the list of available EU conversions.

myCurrentConversion - attribute that indicates which EU conversion is currently in use.

myConvertedValue - attribute that contains the EU converted value.

myMaxConversions - attribute that indicates the maximum number of conversions.

myNewCoefficients - attribute that contains the new coefficients entered by the user.

Convert - operation that initiates the raw to EU conversion.

SelectConversion - operation that allows the user to select the active conversion.

SetConversion - operation that sets the conversion coefficients.

AdjustCurrentCoefficients - operation that initiates the adjustment of coefficients.

FtTlCos - this class represents the arithmetic cosine function.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove a value from a stack and evaluate the cosine of that value.

myQuality - this member variable holds the quality information for an operator.

FtTlDecode - abstract class that is used to provide the decoding algorithm.

myDecodeValue - attribute that contains the decoded value.

DecodeValue - operation that retrieves the decoded value.

FtTlDecom - class that represents the telemetry decommutation process

myNumPacketMap - attribute that indicates the number of packet maps.

myPacketMap - attribute that is an array of packet maps.

Decom - operation that finds the correct packet map to use.

ReadDatabase - operation that reads into memory the packet maps that are stored in the file.

WriteDatabase - operation that writes the current packet maps into the file.

FtTlDecomContextSwitch - class that represents a switch that is used to change the context of the context dependent associated class.

myRawMap - attribute that indicates the decommutation map to use and is determined by the context switch.

myEdu - attribute that represents the Edu that will be sent to the decommutation process.

myParameterTable - attribute that represent the parameter table that will be sent to the decommutation process.

Decom - operation that initiates decommutation when the correct context switch is found.

FtTlDeltaLimit - class that represents an analog delta limit.

myDelta - attribute representing the maximum change allowed between consecutive parameter samples.

myPreviousRawValue - attribute that contains the value of the previous sample of the parameter. Checked against the current value to detect a delta violation.

myStatus - attribute that contains the status of the delta limit check.

mySenseInterval - attribute that represents the current delta-limit sense interval. This attribute determines how often an event message is generated for a continuous delta-limit condition.

mySenseCount - attribute that indicates how many times the parameter has consecutively remained out of limits. This attribute will be compare against the sense interval.

Check - operation that performs the delta limit check.

FtTlDerivedTelemetryMap - this class represents the methods to use when deriving telemetry values from other telemetry points.

Initialize - this member function will, among other things, initialize a pointer to the Parameter Table and some other neat initialization things.

Decom - this member function is responsible for initiating the decommutation of derived telemetry points.

myParameterTablePtr - this member variable points to the parameter table.

myEquations - this member variable points to the equations that are used to derive telemetry points.

myCurrentTime - this member variable is the current time that is used to check against for re-assembling equations.

myWorkSpace - this member variable is a pointer to the stack for evaluating equations.

FtTlDiscreteParam - class that corresponds to the discrete parameter type. Discrete parameters may be boundary limit checked and delta limit checked.

myLimitSet - attribute that points to the limit set associated with the parameter.

myLimitStatus - attribute that indicates the range limit status.

myDeltaLimit - attribute that points to the delta limits associated with the parameter.

myDeltaStatus - attribute that indicates the delta limits check status.

UpdateValue - operation that will update that parameter from the raw values.

FtTlDivide - this class represents the arithmetic division operator.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove two values from a stack and divide the first by the second.

myQuality- this member variable holds the quality information for an operator.

FtTlEdu - class that represents a received EDU. It reads the EDU data from EDOS or DMS interface, and forwards the EDU to be decommed.

myPacketSeqNo - attribute that indicates the sequence number of the packet.

myPacketApid - attribute that indicates the packet identification.

myExpectedPacketApid - attribute that indicates the expected application identification of the packet.

myPacketLength - attribute that indicates the length in bytes of the packet.

myExpectedPacketLength - attribute that indicates the expected packet length.

myPacketScTime - attribute that indicates the spacecraft time of the packet.

myHeaderFlag - attribute that indicates if archiving is on.

myArchiveFlag - attribute that indicates if the Edu header in present.

GetCriticalInfo - operation that gets the packer sequence number, the APID, and the packet spacecraft time.

Verify - operation that checks that the critical information was received.

ReceiveData - operation that gets the Edu.

SetArchiveFlag - operation that sets the archive flag.

GetArchiveFlag - operation that returns the archive flag.

SetHeaderFlag - operation that sets the header flag.

GetHeaderFlag - operation that returns the header flag.

GrabPacketDataBits - operation that gets the data bits and sets the data pointer to the location of the source data.

FtTlElement - this is an abstract base class that represents all of the possible pieces of an equation.

CheckQuality - this member function will check the quality of an element.

Operate - this member function will perform the desired function for the specific type of element that is instantiated.

Initialize - this member function will initialize anything that will be needed for a specific element.

myQuality - this member variable is the quality information for this element.

myValue - this member variable is the value of this element.

FtTlEqual - this class represents the logical equality operator.

CheckQuality - this member function checks the quality of an operator.

Operate - this member function removes two values from a stack and compare them.

Initialize - this member function initializes anything that is needed for an element.

myValue - this member variable holds the value of the element.

myQuality - this member variable holds the quality information for an operator.

FtTlEquation - this class represents the equation that is used to derive a parameter.

IsDue - this member function returns a flag that is used to determine whether it is time to update the parameter associated with the equation.

IsEnabled - this member function returns a flag that is used to determine whether the decommutation of the parameter associated with an equation is enabled.

Calculate - this member function calculates the value of the parameter that is associated with an equation.

GetRate - this member function gets the update interval.

SetNextUpdate - this member function sets the next update time for an equation.

SetUpdateRate - this member function sets the update rate for an equation in seconds.

SetEnabledFlag - this member function sets the Enabled flag for an equation.

myEnabledFlag - this member variable flags whether an equation is enabled or not.

myRawValue - this member variable is the raw value of the derived parameter.

myQuality - this member variable contains the quality information for the equation.

myUpdateInterval - this member variable is the rate of update for the derived parameter.

myNextUpdateTime - this member variable is used to determine when to update a derived parameter.

FtTlEuConversion - abstract class that represents an EU conversion.

mySelectedFlag - attribute indicating whether the user has selected the conversion. This attribute provides a mechanism for the user to lock the conversion. The selected flag overrides any context dependence conversion switching.

myContextSwitch - attribute that, if present, provides an association between the conversion and a context switch. Context switching is based upon the value of a discrete parameter.

myNewCoefficients - attribute that holds the user specified coefficients.

myEuValue - attribute that holds the current EU converted value of the conversion.

myStatus - attribute that indicates the status of the EU conversion process.

Convert - operation that converts the raw value to an EU value.

AjustCurrentCoefficients - operation that initiates the adjustment of coefficients.

SetCoefficient - operation that allows the setting of coefficients.

FindConversion - operation that finds the conversion using the context dependent parameters.

FtTlExponentialConversion - class that represents a type of EU conversion. The exponential conversion uses the following equation:

$y = C_0 + C_1 e^{(c_2 x)}$.

myCoefficients - attribute that contains the list of coefficients used in the calculation of the EU.

myNewCoefficients - attribute that contains the list of coefficients specified by the user.

Convert - operation that converts the raw value to an EU.

AdjustCurrentCoefficients - operation that adjusts the current coefficients.

SetCoefficient - operation that sets the coefficients.

FtTlGreater - this class represents the logical greater than operator.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove two values from a stack and compare them.

myValue - this member variable holds the value of the element.

myQuality - this member variable holds the quality information for an operator.

FtTlGreaterOrEqual - this class represents the logical greater than or equal to operator.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove two values from a stack and compare them.

myQuality - this member variable holds the quality information for an operator.

FtTlLess - this class represents the less than operator.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove two values from a stack and compare them.

myQuality - this member variable holds the quality information for an operator.

FtTlLessOrEqual - this class represents the logical less than or equal to operator.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove two values from a stack and compare them.

myQuality - this member variable holds the quality information for an operator.

FtTlLimitSet - class that represents the set of limits available to an associated analog parameter.

myCurrentGroup - attribute indicating which limit boundary group is currently selected.

mySenseInterval - attribute that represents the current out-of-limits sense interval. This attribute determines how often an event message is generated for a continuous out-of-limit condition.

mySenseCount - attribute that indicates how many times the parameter has consecutively remained out of limits. This attribute will be compare against the sense interval.

myBoundaryGroups - attribute that contains a list of available limit boundary groups.

myMaxGroups - attributes that indicates the maximum number of boundary groups.

myStatus - attributes that indicates the current processing status.

Check - operation that initiates delta and boundary limit checking.

GetLimits - operation that returns a set of limits.

SetLimits - operation that sets the limits for a parameter.

AdjustBoundaryGroup - operation that initiates the adjustment of the limits.

SelectBoundaryGroup - operation that allows the user to select a boundary group.

FtTlLimits - class represents the range limit high and low values.

myLoValue- attribute that holds the low limit value.

myHiValue - attribute that holds the high limit value.

myValue - attribute that holds the value that is being limit checked.

myStatus - attribute that holds the type of limit violation that occurred.

myLimitType - attribute that holds the limit type of the class.

Check - operation that checks for limit violations.

SetHiValue- operation that sets the high value for the limits.

SetLoValue - operation that sets the low value for the limits.

AdjustBoundaryGroup - operation that allows the user to adjust the limit range.

FtTlLineConversion - class that represents a type of EU conversion. The line conversion uses linear interpolation of the raw analog parameter value. Up to fifteen contiguous line segments of increasing value may be associated with the conversion.

myLineSegments - attribute that contains the list of line segments used in the calculation of the EU.

myCurrentLineSegment - attribute that indicates the line segment to use for the EU conversion.

Convert - operation that initiates the raw to line segment EU conversion.

FtTlLineSegment - class that represents a line segment used during the line segment interpolation process.

myX1 - attribute holding the x coordinate of the start point of the line segment.

myY1 - attribute holding the y coordinate of the start point of the line segment.

myX2 - attribute holding the x coordinate of the end point of the line segment.

myY2 - attribute holding the y coordinate of the end point of the lint segment.

myActiveFlag - derived attribute indicating whether the slope and intercept for the line segment have been previously calculated.

myCurrentSegmentFlag - attribute that indicates the line segment in use for the EU conversion.

mySlope - attribute representing the calculated slope of the line segment.

myIntercept - attribute representing the calculated y-intercept of the line segment.

myDecodedValue - attribute representing the decoded value to be EU converted.

ComputeSlopeIntercept - operation that calculates the slope and y-intercept of the line segment if they have not been previously calculated.

Convert - operation that calculates the raw to line segment EU conversion.

Check - operation that determines the correct line segment to use.

FtTlMultiply - this class represents the arithmetic multiplication operator.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove two values from a stack and multiply them together.

myQuality - this member variable holds the quality information for an operator.

FtTlNegate - this class represents the arithmetic unary minus operator.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove a value from a stack and apply a unary minus.

myQuality - this member variable holds the quality information for an operator.

FtTlNot - this class represents the logical NOT operator.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove a value from a stack and apply a NOT.

myQuality - this member variable holds the quality information for an operator.

FtTlNotEqual - this class represents the logical inequality operator.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove two values from a stack and compare them.

myQuality - this member variable holds the quality information for an operator.

FtlOperator - this class is an abstract class that represents operators used in FtTlEquations for derived telemetry.

CheckQuality - this member function will check the quality.

Operate - this member function will remove a value(s) from a stack and operate on them.

myQuality - this member variable holds the quality information for an operator.

FtTlOr - this class represents the logical OR operator.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove two values from a stack and OR them.

myQuality - this member variable holds the quality information for an operator.

FtTlPacketMap - class that represents the packet maps for decom. It begins the processing of parameter maps.

myParamMap - attribute that is an array of parameter maps.

myNumPacketParams - attribute that indicates the number of parameters in the packet.

myEdu - attribute that represents the Edu that will be sent to the decommutation process.

myParameterTable - attribute that represents the parameter table that will be sent to the decommutation process.

Decom - operation that initiates decommutation for the parameter maps.

FtTlParameter - class that maintains the parameter values for a single parameter.

myDecomFlag - attribute that indicates whether this parameter should be decommutated.

myActiveFlag - attribute that indicates whether the parameter is currently active and being updated. For sampled telemetry, this indicates that the parameter is being decommutated. For derived telemetry, this indicates that the parameter is being calculated. In the event of a loss of data or a data dropout, this flag is set to reflect a static condition.

myStaticToggle - attribute that indicates if the parameter is static.

myNumberofValues - attribute that indicates the number of values for the parameter.

ValuesServedFlag - attribute that indicates the amount of values that have been served for the parameter.

GetActiveFlag - operation that retrieves the active flag.

SetActiveFlag - operation that sets the active flag.

GetDecomFlag - operation that retrieves the decommutation flag.

SeDecomFlag - operation that sets the decommutation flag.

GetLimits - operation that retrieves the limit set.

SetLimits - operation that sets the range limits for this parameter.

SetDeltaLimit - operation that sets the delta limits for this parameter.

SetQuality - operation that sets the quality flag.

SetDecodeValue - operation that sets the decoded value.

GetDecomFlag - operation the retrieves the decommutation flag.

SetDecomFlag - operation that sets the decommutation flag.

SetValuesServedFlag - operation that sets the values served flag.

SelectDecom - operation that turns decommutation on or off for a parameter.

ConvertValue - operation that gets the converted value.

Update - operation that initiates parameter processing (i.e., limit checking, EU conversion).

UpdateValue - operation that performs inherited class specific updates.

FtTlParameterContextSwitch - class that represents a switch that is used to issue an event message that the quality of the parameter is bad or the parameter has been marked static.

myStatus - attribute that indicates the current processing status.

Compare- operation that checks the quality of the parameter.

FtTlParameterTable - class that maintains the values for all of the parameters.

myParameter - attribute that is the table of all of the parameters.

myStatus - attribute that indicates the static status of the parameter.

Update - operation that updates the parameter table element indicated by the pid.

StaticCheck - operation that performs a static check on all of the parameters in the table.

GetActiveFlag - operation that retrieves the active flag.

SetQuality - operation that sets the quality of the parameter.

GetQuality - operation that retrieves the quality of the parameter.

GetConvertedValue - operation that retrieves the converted value of the parameter.

GetCurrentValue - operation that retrieves the current value of the context parameter.

GetLimits - operation that retrieves the limit set.

GetDecodedValue - operation that retrieves the decoded value of the parameter.

GetRawValue - operation that retrieves the raw value of the parameter.

ReadDatabase - operation that reads the parameter object stored in the database.

WriteDatabase - operation that writes the current parameter into the configuration database.

FtTlParameterValues - class that is used to maintain all values that are unique to a single instance of parameter.

myRawValue - attribute that represents the raw bit value received.

myPid - attribute that represents the parameter identification.

myDecodedValue - attribute that contains the decoded value.

myConvertedValue - attribute that contains the converted value.

myMnemonic - attribute that contains the mnemonic.

myFirstBitOffset - attribute that contains the first bit offset used to calculate the time tag.

myStatus - attribute that contains the status.

myQuality - attribute that contains the quality.

mySubSystemId - attribute that contains the subsystem identification.

GetRawValue - operation that retrieves the raw value of the parameter.

GetDecodedValue - operation that retrieves the decoded value.

GetConvertedValue - operation that retrieves the converted value.

GetSubsystemId - operation that retrieves the subsystem identification.

GetQuality - operation that retrieves the quality.

FtTlParamMap - class that represent the parameter maps.

myEdu - attribute that represents the Edu that will be sent to the decommutation process.

myParameterTable - attribute that represents the parameter table that will be sent to the decommutation process.

Decom - operation that initiates the decommutation process.

FtTlParamOperand - this class represents parameter values used in FtTlEquations for derived telemetry.

CheckQuality - this member function will check the quality.

Operate - this member function will place a parameter value on the stack.

Initialize - this member function will initialize a link to the Parameter Table.

myValue - this member variable holds the value of the parameter.

myQuality - this member variable holds the quality information for the parameter.

myPid - this member variable is the PID for the associated parameter.

myParameterTablePtr - this member variable is a pointer to the parameter table.

FtTlPolyConversion - class that represents a type of EU conversion. The polynomial conversion uses the following equation:

$y = C_0 + C_1x + C_2x^2 + ... C_7x^7.$

myCoefficients - attribute that contains the list of coefficients used in the calculation of the EU.

myNewCoefficients - attribute that contains the list of coefficients specified by the user.

Convert - operation that converts the raw value to an EU value.

AdjustCurrentCoefficients - operation that adjusts the current coefficients.

SetCoefficient - operation that allows setting of the coefficients.

FtTlRawMap - class that represents the map for decom. It initiates getting the raw value and updating the parameter table.

myPid - attribute that represents the parameter ID.

myRawValue - attribute that contains the raw value.

myTargetParameter - attribute that is being filled by the decommutation process.

myComponentMap - attribute that is an array of component maps.

myFirstBit - attribute that is the first bit.

myFirstBitOffset - attribute that is the first bit's offset.

GetPid - operation that obtains the parameter ID.

Decom - operation that gets the raw value.

FtTlSin - this class represents the arithmetic sine function.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove a value from a stack and evaluate the sine of that value.

myQuality - this member variable holds the quality information for an operator.

FtTlStatus - .this class represents all of the parameter's statuses.

myStatus - this member variable holds the current status.

myStatusType - this member variable is the current type of status.

Set - this operation sets the status types to send to the parameter server.

Get - this operation gets the requested status value.

FtTlSubtract - this class represents the arithmetic subtraction operator.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove two values from a stack and subtract them.

myQuality - this member variable holds the quality information for an operator.

FtTlTan - this class represents the arithmetic tangent function.

myQuality - this member variable holds the quality information for an operator.

CheckQuality - this member function will check the quality of an operator.

Operate - this member function will remove a value from a stack and evaluate the tangent of that value.

FtTlTelemetryController - class that is responsible for controlling an instance of the telemetry subsystem process. This class receives and processes configuration adjustment requests.

myParameterTable - attribute that represents the parameters and their values.

myDecom - attribute that indicates an instance of decom.

myDerivedTelemetryMap - attribute that represents a derived telemetry map.

myConfigRequest - attribute that represents a configuration adjustment request.

mySCStateCheck - attribute that represents the state check request.

ProcessRequest - operation that processes a configuration request.

Initialize - operation that initializes attributes and interfaces.

Run - operation that runs the decommutation controller process.

Shutdown - operation that shuts down the decommutation controller process.

3.3 Memory Dump

The Memory Dump Subsystem provides the capability to collect and store the contents of downlinked spacecraft or instrument computer memory dumps. It will detect and notify the user at the start and completion of a computer memory dump. All memory dump data will be stored in a file.

3.3.1 Memory Dump Context

The Memory Dump Subsystem context diagram shown in Figure 3.3-1 depicts the data flows between the FOS Memory Dump Subsystem and external ground system as well as EOC internal components. Descriptions of the data flows are summarized for each component:

EDOS: The EDOS forwards telemetry to Memory Dump Subsystem via EDOS Data Units(EDUs). Each EDU contains a reconstructed CCSDS telemetry packet, quality information, and time stamp. The packetized message transports spacecraft or instrument computer memory dumps.

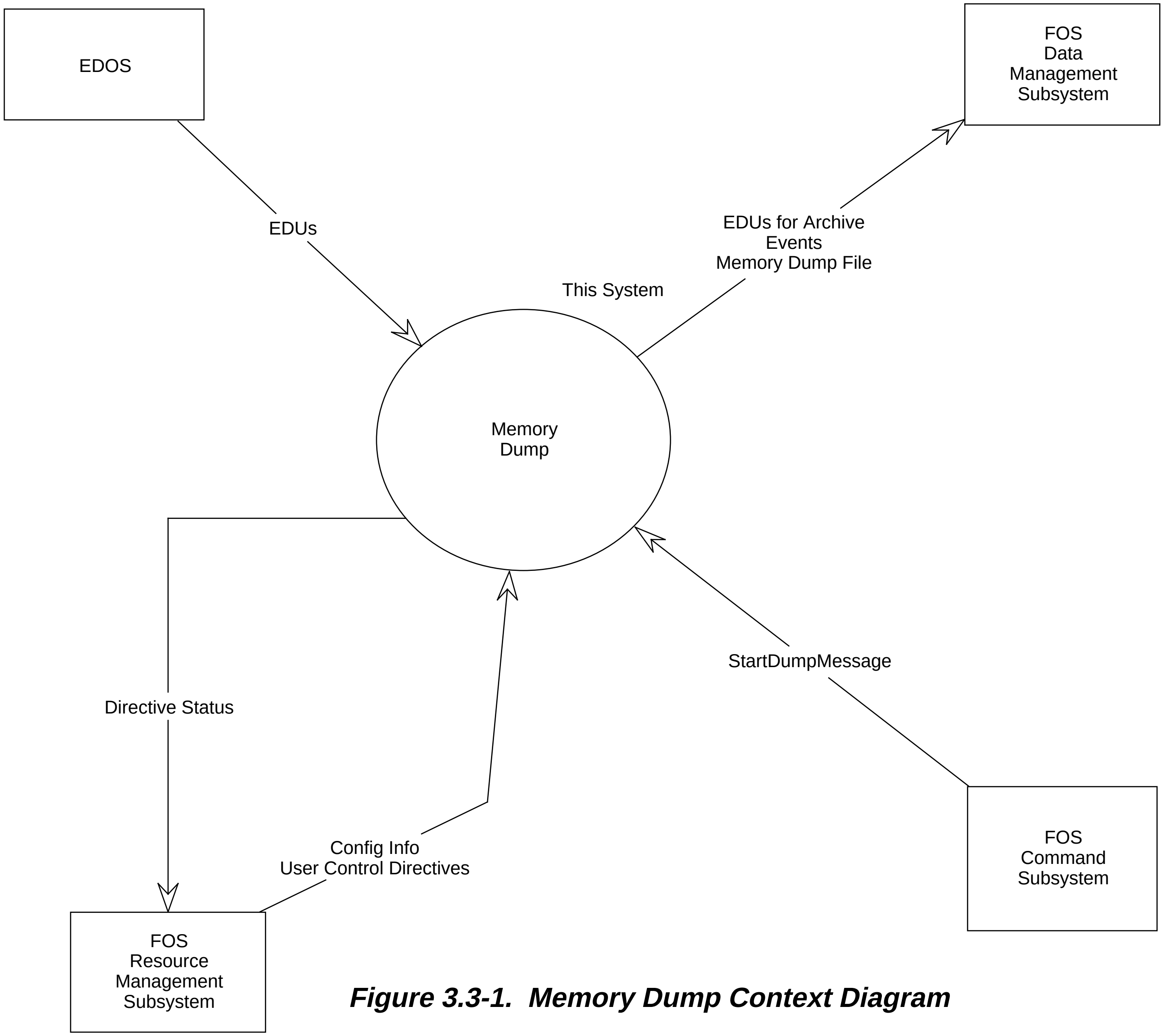

Figure 3.3-1. Memory Dump Context Diagram

FOS Data Management Subsystem: During a memory dump session, all EDUs received by the MemoryDumpSubsystem (if archiving is enabled) and memory dump events are forwarded to the Data Management Subsystem for storage and processing. At the end of a memory dump session, the file containing the memory dump EDUs is forwarded to the Data Management Subsystem for storage

FOS Resource Management Subsystem: The Resource Management Subsystem supplies configuration information required by Memory Dump Subsystem for memory dump processing. This data includes EDOS and Command Management Subsystem communication channels, and user configuration requests.

FOS Command Management Subsystem: The Command Management Subsystem supplies the StartDumpMessage to the Memory Dump Subsystem to initialize a memory dump session. The StartDumpMessage includes information on the source and the size of the memory dump.

3.3.2 Memory Dump Interfaces

Table 3.3-1 Memory Dump Interfaces

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Memory Dump Configuration Proxy	FtTlDumpConfig	Provides for configuring and controlling of a memory dump process	TLM	RMS	At initialization of a telemetry process and upon user directive
EDOS interface	FtTlEdu	Provides EDUs for memory dump	TLM	TLM	Every EDU
Telemetry Archiver interface	FdArTlmArchProxy	Archives EDUs	DMS	TLM	Every EDU
Memory Dump Start Message interface	FoGnTlmProxy	Provides the StartDumpMessage	CMD	TLM	At the start of each memory dump session.

3.3.3 Memory Dump Object Model

The Memory Dump Object Model is depicted in Figure 3.3-2. The following section describes the objects in the Memory Dump Object Model.

FtTlDumpController class is the controller of the memory dump process. This class configures the process and controls the different memory dump states.

FoGnTlmSourceIF class is the telemetry source interface. This class receives the data and performs the communications layer interface.

FtTlEdu class obtains and verifies the critical information from the EDU. If archiving is enabled, this class sends the EDUs to be archived by DMS.

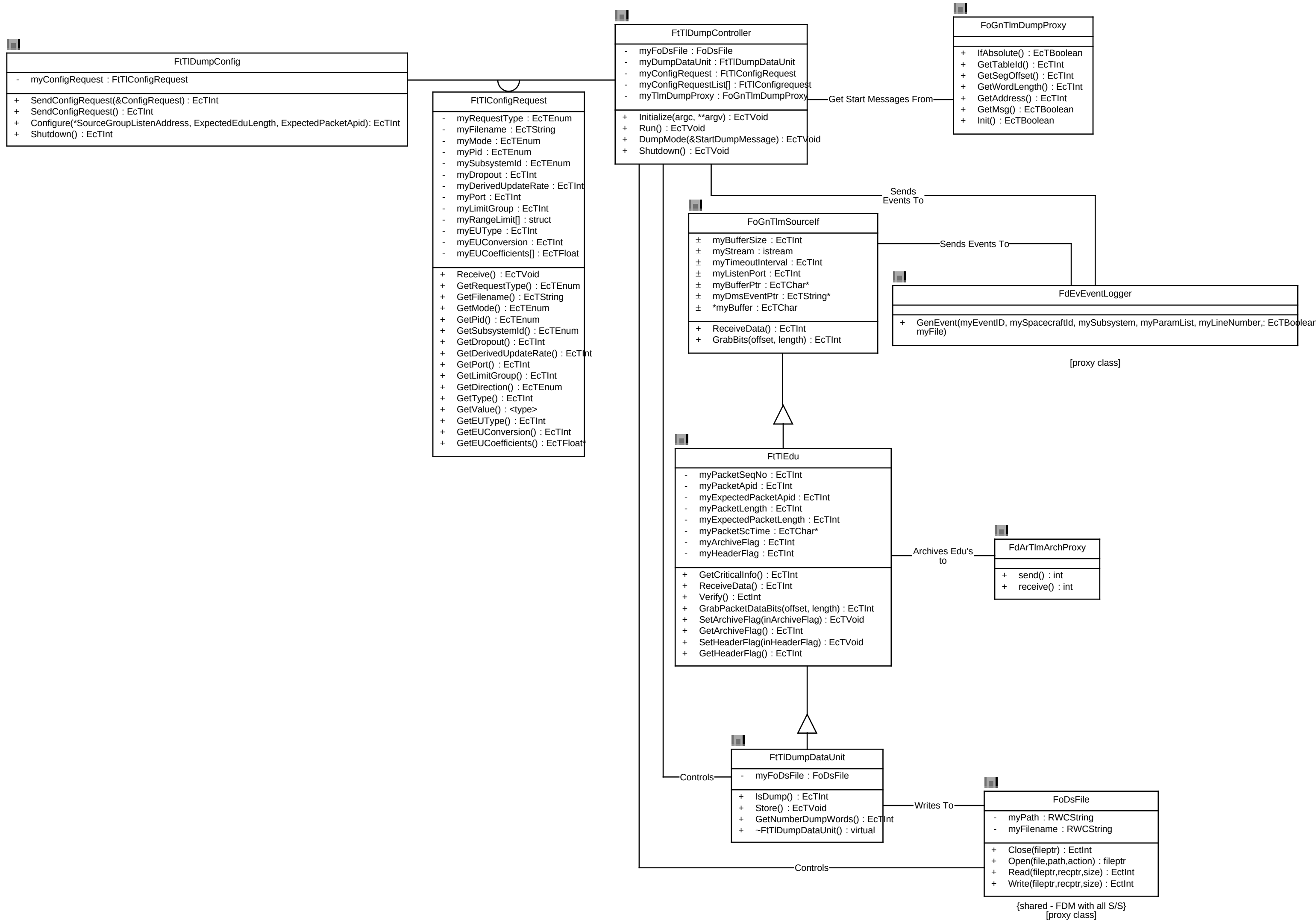

Figure 3.3-2. Memory Dump Object Model

FtTlDumpDataUnit inherits from the FtTlEdu class which inherits from the FoGnTlmSourceIF class. This class provides dump specific extraction and verification capabilities.

FtTlDumpConfig is the proxy to allow RMS to control the memory dump process.

FtTlConfigRequest is the link class used to carry the information from the FtTlDumpConfig proxy to the memory dump process.

FoGnTlmDumpProxy is the proxy provided by CMD which gets the StartDumpMessage to the dump process. This message initiates a dump session.

FdEvEventLogger is the proxy provided by DMS which handles event messages.

FdArTlmProxy is the proxy provided by DMS which accepts and archives EDUs.

FoDsFile is the utility provided by DMS which performs file I/O and manipulations.

3.3.4 Memory Dump Dynamic Model

The Memory Dump Subsystem is dynamically modeled in the state transition diagram of FtTl-DumpController (Figure 3.3-3). The following scenarios and event trace diagrams detail the transition between the two main states (i.e. the Awaiting Message State and the Dump Mode State). Combined, these sequence of events describe how a dump session captures a memory dump. The following scenarios are described in this section:

Awaiting Message State Scenario

Dump Mode State Scenario

3.3.4.1 Awaiting Message State Scenario

3.3.4.1.1 Awaiting Message State Scenario Abstract

The purpose of "Awaiting Message State Scenario" is to initialize a memory dump session once it gets the message to start. The event trace for this scenario can be found in Figure 3.3-4.

3.3.4.1.2 Awaiting Message State Summary Information

Interfaces:

CMD

Stimulus:

FtTlDumpController receives a StartDumpMessage from the CMD interface.

Desired Response:

The memory dump process initiates the dump session and goes into Dump Mode State.

Pre-Conditions:

The memory dump process has been initialized and is in the Awaiting Message State.

Post-Conditions:

The memory dump process is in Dump Mode and is ready to receive additional EDUs.

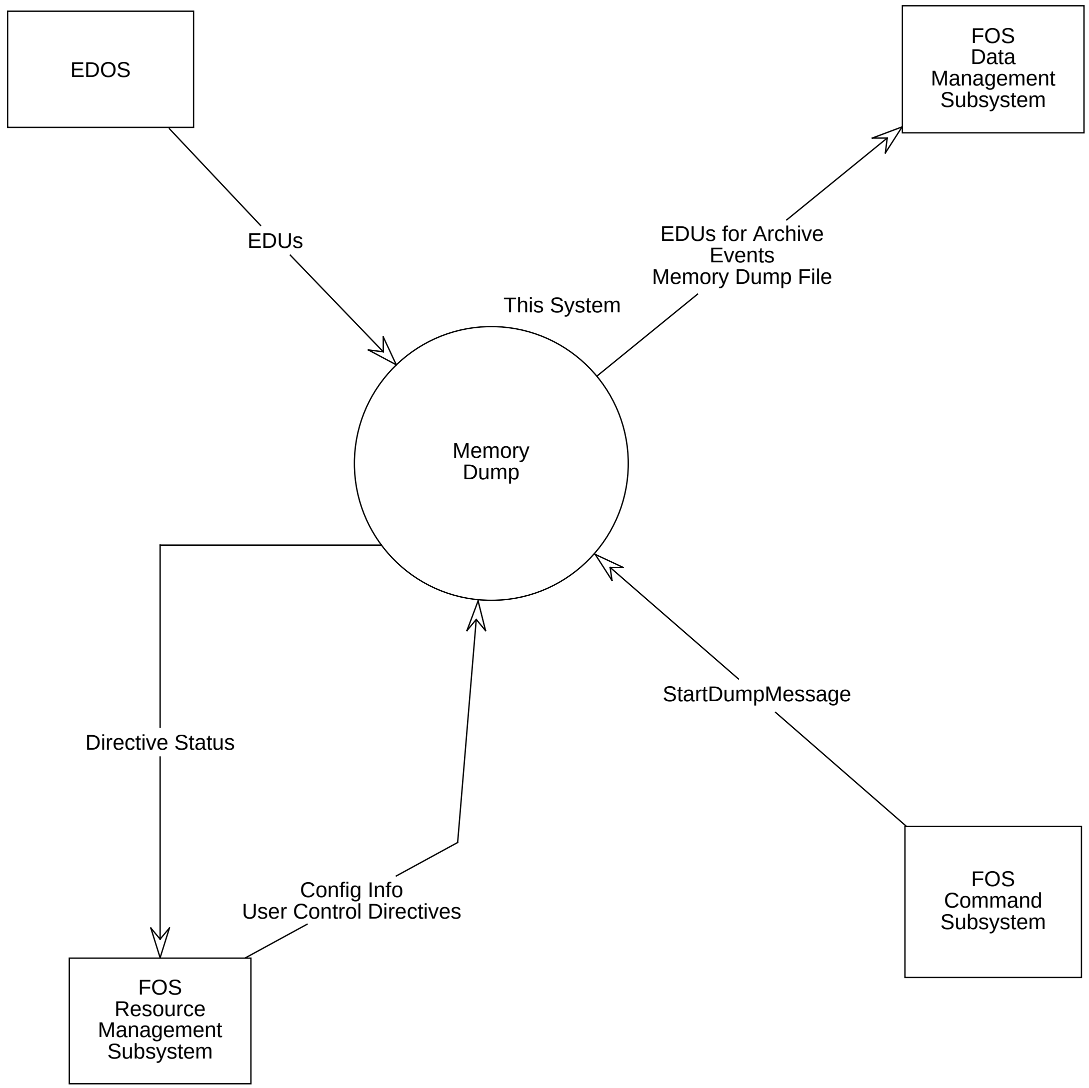

Figure 3.3-3. Memory Dump State Transition Diagram

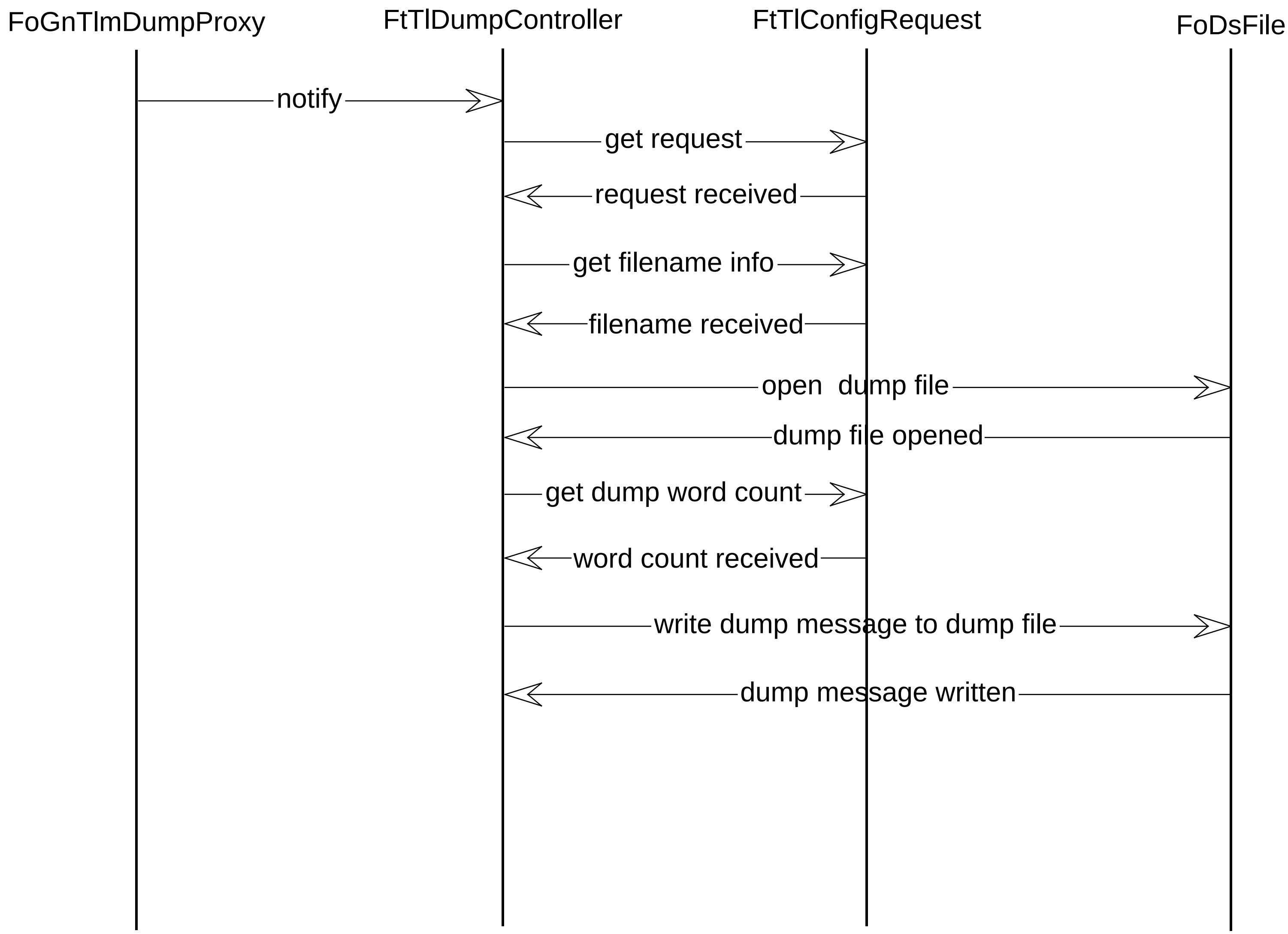

Figure 3.3-4. Awaiting Message State Event Trace

3.3.4.1.3 Awaiting Message State Scenario Description

FoGnTlmDumpProxy initiates the memory dump session by notifying FtTlDumpController that a StartDumpMessage has arrived. FtTlDumpController calls FtTlConfigRequest to receive the request. FtTlDumpController calls FtTlConfigRequest to get information required to generate the dump storage file name. FtTlDumpController uses the information in the StartDumpMessage to generate a DumpFileName and then calls FoDsFile to open that file. FtTlDumpController then calls FoDsFile to write out the StartDumpMessage to the dump storage file.

FtTlDumpController calls FtTlConfigRequest to get the DumpWordCount and initializes the DumpWordCount with information from the StartDumpMessage. FtTlDumpController then waits in Dump Mode for the memory dump EDUs to arrive.

3.3.4.2 Dump Mode State Scenario

3.3.4.2.1 Dump Mode State Scenario Abstract

The purpose of "Dump Mode State Session Scenario" is to perform the actual memory dump and to detect and terminate the end of a memory dump session. This scenario starts in Dump Mode State and accepts EDUs and determines if they are memory dump EDUs. All memory dump EDUs are written out to the dump storage file. Once all memory dump EDUs have been received (as indicated by the DumpWordCount) an event message is generated and the memory dump session is completed. The event trace for this scenario can be found in Figure 3.3-5.

3.3.4.2.2 Dump Mode State Summary Information

Interfaces:

DMS

EDOS

Stimulus:

Memory dump EDUs arrive from EDOS.

Desired Response:

All memory dump EDUs are written out to the dump storage file.

Pre-Conditions:

The memory dump process is in Dump Mode and is ready to receive EDUs.

Post-Conditions:

All memory dump EDUs for this memory dump session have been received.

3.3.4.2.3 Dump Mode State Scenario Description

When an EDU arrives, FtTlDumpController calls FtTlDumpDataUnit to receive the EDU. FtTlDumpController then calls FtTlDumpDataUnit to determine if the EDU is a memory dump EDU. When this is a memory dump EDU and if it is the first memory dump EDU detected in this dump session FtTlDumpController calls FdEvEventLogger to send a "Beginning Dump" event message. For all memory dump EDUs FtTlDumpController calls FtTlDumpDataUnit to verify the EDU. When the EDU passes verification FtTlDumpController calls FtTlDumpDataUnit to store the memory dump EDU. FtTlDumpDataUnit calls FoDsFile to store the memory dump EDU. FtTlDumpController decrements the DumpWordCount by the number

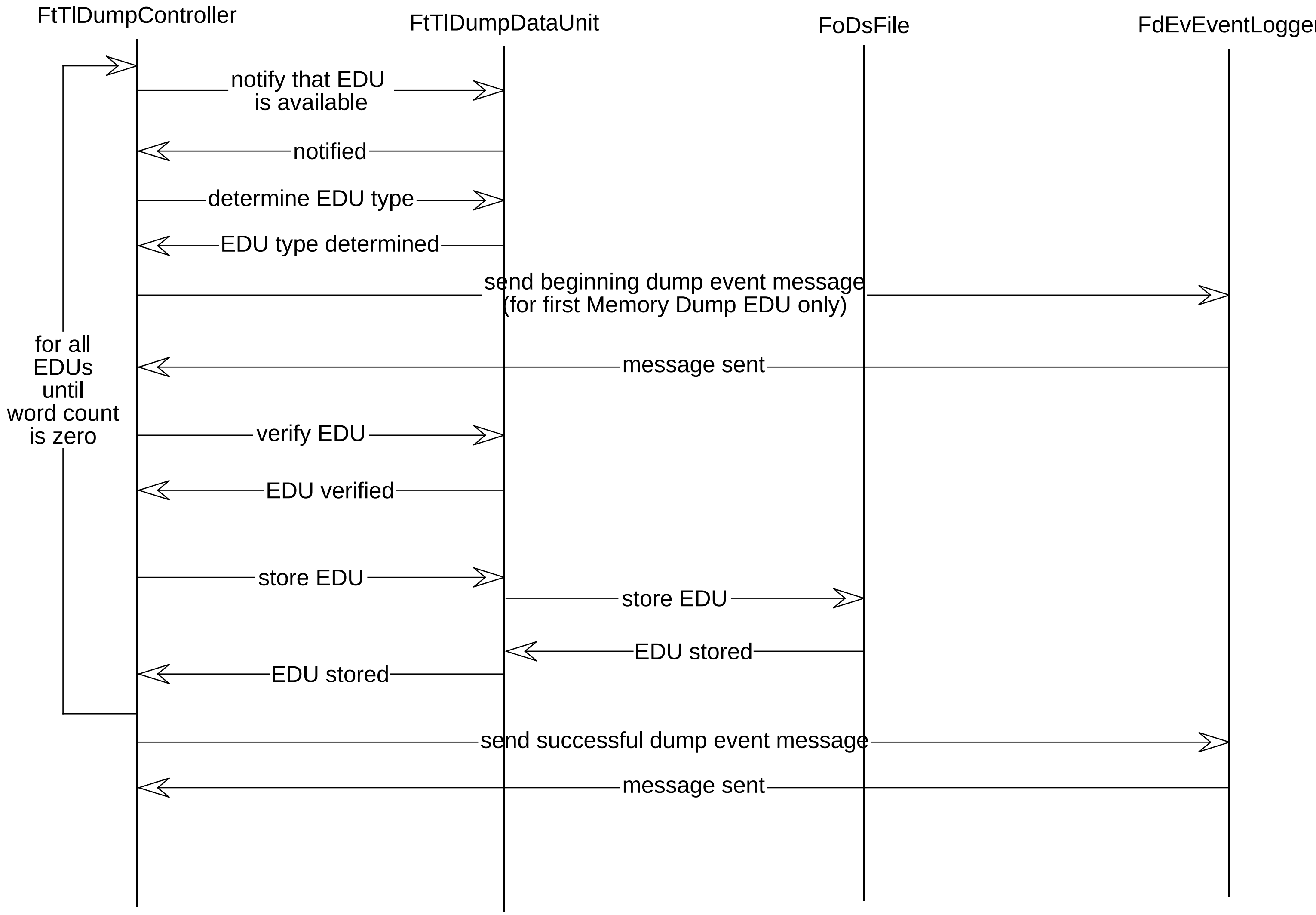

Figure 3.3-5. Dump Mode State Event Trace

of words dumped in the memory dump EDU. FtTlDumpController continues in Dump Mode until the DumpWordCount goes to zero. When the DumpWordCount is zero FtTlDumpController calls FdEvEventLogger to send an "Successful Dump Completed, Total words dumped = XXX" Event Message.

3.3.5 Memory Dump Data Dictionary

FdArTlmArchProxy - class that is a proxy from DMS that archives EDUs.

FdEvEventLogger - class that is a proxy from DMS for logging events.

FoDsFile - class that is the proxy provided by DMS which performs file I/O and manipulations.

FoGnTlmDumpProxy - class that is the proxy provided by CMD which gets the StartDumpMessage to the dump process.

FoGnTlmSourceIF - class that initiates initialization of connections through a port. It receives a data stream, checks for errors and writes the data to a buffer.

> **myBuffer** - attribute that stores the data.
>
> **myBufferPtr** - attribute that points to the location of the data in the data buffer.
>
> **myBufferSize** - attribute that indicates the size of the buffer.
>
> **myDmsEventPtr** - attribute that points to an event message when an error has occurred.
>
> **myListenPort** - attribute that represents the listening port number.
>
> **myStream** - attribute that represents the data stream.
>
> **myTimeoutInterval** - attribute that represents the time interval between data.
>
> **GrabBits** - operation that extracts bits from the buffer.
>
> **ReceiveData** - operation that fills the buffer with data.

FtTlConfigRequest - class that corresponds to configuration update requests.

> **myDerivedUpdateRate** - attribute that contains the rate of updating derived parameters.
>
> **myDropout** - attribute that contains the dropout interval.
>
> **myEUCoefficients** - attribute that contains the EU coefficients.
>
> **myEUConversion** - attribute that contains the EU conversion indicator.
>
> **myEUType** - attribute that contains the EU conversion type.
>
> **myFileName** - attribute that contains the filename used for a WriteDatabase or ReadDatabase request.
>
> **myLimitGroup** - attribute that contains the limit group to set.
>
> **myMode** - attribute that contains the on or off mode used for archiving or selective decom.
>
> **myPid** - attribute that contains the parameter identification.
>
> **myPort** - attribute that contains the input telemetry port.
>
> **myRangeLimit** - attribute that contains the range limit information.
>
> **myRequestType** - attribute that contains the type of request.
>
> **mySubsystemId** - attribute that contains the subsystem identification.

GetDerivedUpdateRate - operation that returns the derived update rate.

GetDirection - operation that returns the range limit direction.

GetDropout - operation that returns the dropout interval.

GetEUCoefficients - operation that returns the EU coefficients.

GetEUConversion - operation that returns the EU conversion.

GetEUType - operation that returns the EU type.

GetFileName - operation that returns the filename.

GetLimitGroup - operation that returns the limit group.

GetMode - operation that returns the mode.

GetPid - operation that returns the parameter identification.

GetPort - operation that returns the telemetry port.

GetRequestType - operation that returns the request type.

GetSubsystemId - operation that returns the subsystem identification.

GetType - operation that returns the range limit type.

GetValue - operation that returns the range limit value.

Receive - operation that receives the data from an external interface.

FtTlDumpConfig - This class acts as the proxy to RMS to communicate with a TLM decom process.

SendConfigRequest - This operation sends a configuration request to telemetry.

Configure - This operation configures the memory dump process with setup information.

Shutdown - This operation sends a shutdown message to the memory dump process.

myConfigRequest - This attribute represents the memory dump configuration request.

FtTlDumpController - This class is responsible for controlling an instance of the Memory Dumping process. The Dump Controller checks for inputs to the Dump process and then controls the state of the process.

Initialize - This operation sets up the initial state of the memory dump process.

Run - This routine controls the different states of the memory dump process.

DumpMode - This routine is called when the memory dump process is entering Dump Mode. It will detect the start of a memory dump, do the dump, and respond to Dump Messages.

Shutdown - This routine does any required cleanup before the memory dump process exits.

myFoDsFile - This attribute is the file object which we are writing the dump Edu's to.

myDumpDataUnit - This attribute is the dump data unit object which contains the dump Edu's.

myConfigRequest - This attribute represents the configuration request .

myConfigRequestList - This attribute represents the configuration request list.

myTlmDumpProxy - This attribute is the proxy to CMD which supplies the Start Dump Message.

FtTlDumpDataUnit - This class is used to deal with Memory Dump Data

IsDump - This operation returns true if the current EDU is a memory dump EDU.

Store - This operation stores the current EDU into a file.

GetNumberDumpWords - This operation returns the number of memory dump words in the current EDU.

myFoDsFile - This attribute is the file object which we are writing the dump Edu's to.

FtTlEdu - class that represents a received EDU. It reads the EDU data from EDOS or DMS interface, and forwards the EDU to be decommed.

myPacketSeqNo - attribute that indicates the sequence number of the packet.

myPacketApid - attribute that indicates the packet identification.

myExpectedPacketApid - attribute that indicates the expected application identification of the packet.

myPacketLength - attribute that indicates the length in bytes of the packet.

myExpectedPacketLength - attribute that indicates the expected packet length.

myPacketScTime - attribute that indicates the spacecraft time of the packet.

myHeaderFlag - attribute that indicates if archiving is on.

myArchiveFlag - attribute that indicates if the Edu header in present.

GetCriticalInfo - operation that gets the packer sequence number, the APID, and the packet spacecraft time.

Verify - operation that checks that the critical information was received.

ReceiveData - operation that gets the Edu.

SetArchiveFlag - operation that sets the archive flag.

GetArchiveFlag - operation that returns the archive flag.

SetHeaderFlag - operation that sets the header flag.

GetHeaderFlag - operation that returns the header flag.

GrabPacketDataBits - operation that gets the data bits and sets the data pointer to the location of the source data.

3.4 Spacecraft State Check

The Spacecraft StateCheck assists in back-orbit command verification. It allows the EOC to baseline the current states of the spacecraft, monitor and compare the spacecraft's state with a baseline, and compare the spacecraft's state with it's expected state.

3.4.1 Spacecraft State Check Context

RMS: The RMS subsystem provides the StateCheck process with Command Line Parameters that gives StateCheck the Parameter Server identifier (Fig 3.4-1). The RMS interface

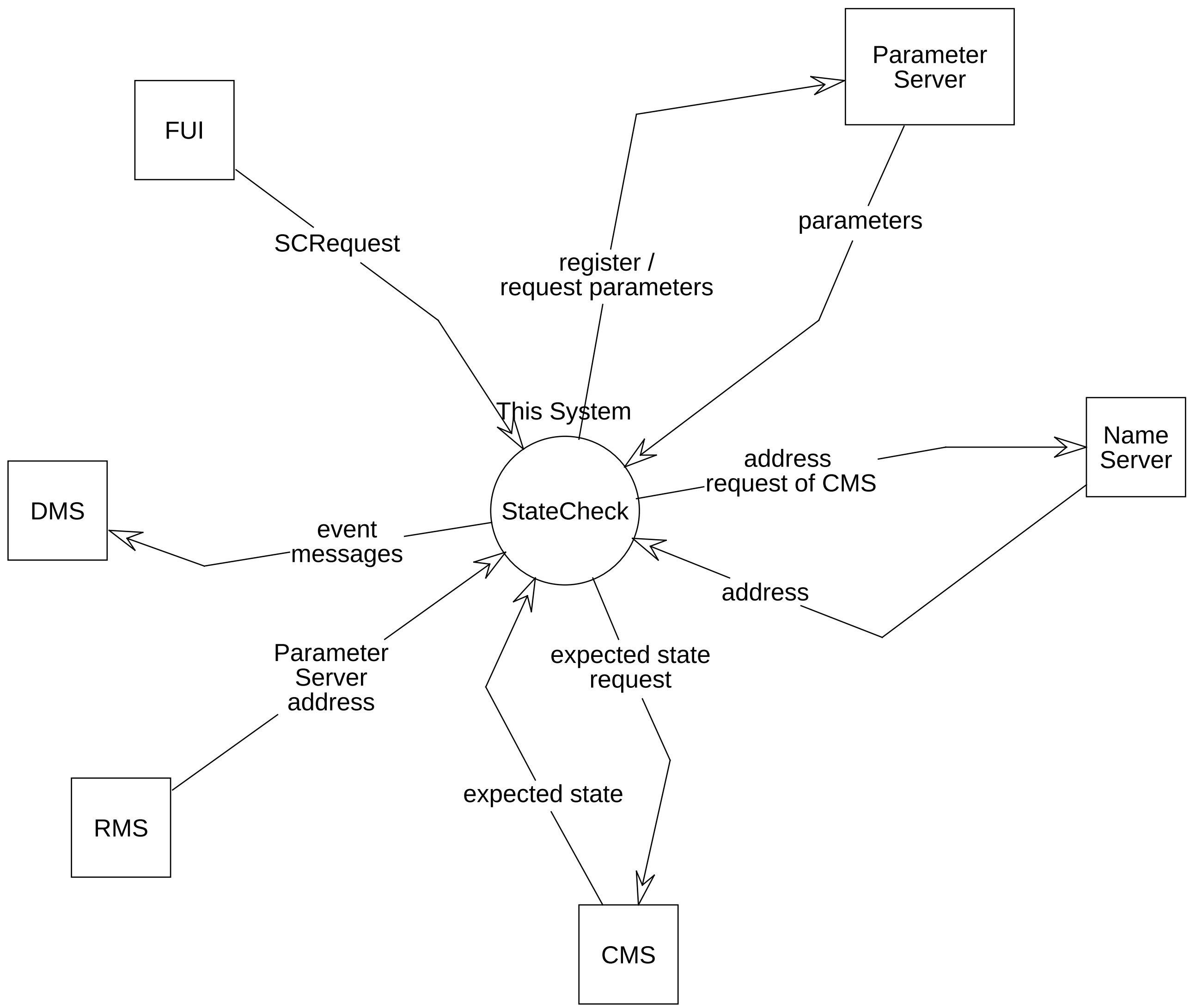

Figure 3.4-1. Spacecraft State Check Context Diagram

is limited to statecheck process creation and therefore will not be seen in the rest of this subsystem.

DMS: The DMS subsystem receives event messages from the StateCheck process when a miscompare is encountered. It also receives a summary event message upon completion of a state check.

FUI: The FUI subsystem sends a request to the StateCheck process with a state check argument. This argument determines if the request is to load, baseline, or perform a state check. A load argument will cause the StateCheck process to get an expected value table from CMS. A baseline causes the StateCheck process to replace the values in the expected value table with current values retrieved from the Parameter Server. A perform state check argument causes the StateCheck process to compare the values in the expected value table with current values retrieved from the Parameter Server.

Parameter Server: The Parameter Server subsystem provides the StateCheck process with current downlink telemetry values from the space craft. The StateCheck process registers with the Server and requests parameters when needed.

NameServer: The Name Server provides the StateCheck process with the network address of CMS.

CMS: The CMS subsystem provides the StateCheck process with an expected value table that will be used during a perform state check argument.

3.4.2 Spacecraft State Check Interfaces

Table 3.4-1. Spacecraft State Check Interfaces (1 of 2)

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Load	StateCheckRequest	Loads the expected state values onto a table.	TLM	FUI	Minimum of once per pass
Baseline	StateCheckRequest	Replaces the expected state table with current downlink tlm.	TLM	FUI	Typically once per pass.
StateCheck	StateCheckRequest	Compares current downlink tlm with the expected state.	TLM	FUI	Minimum of once per pass.
FetchTable	ExpectedStateTable	provides the state check process with a table of expected values.	CMS	TLM	When a state check is called.
Event Generator	FdEvEvent Logger	generates events to DMS	DMS	TLM	When an event message is sent.
Receive Buffer	PsClientIF	Makes a buffer of requested parameters.	Parameter Server	TLM	When state check & baseline requests are made.

Table 3.4-1. Spacecraft State Check Interfaces (2 of 2)

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Register Client	PsClientIF	Register a client as continuous	Parameter Server	TLM	When a receive buffer request is made to parameter server.
NameServer	Directory_Name_Service	Return the network address of requested processes	NameServer	TLM	When the state check process is initialized.

3.4.3 Spacecraft State Check Object Model

FtTlStateCheckController class is the controller of the process. It establishes connections with FUI, CMS and Parameter Server when the StateCheck process is initialized (see Fig 3.4-2). It gets a request from FtTlStateCheckRequest and determines if the request is a load, baseline, or a perform statecheck, then performs the request.

FtTlStateCheckRequest is a class that acts as a proxy to FUI. It gets a request from FUI and can return the request type and argument.

FtTlSCStateCheck will load expected values, baseline or perform a statecheck against expected values. A load will cause FtTlSCStateCheck to get an expected state value table from CMS. A baseline causes FtTlSCStateCheck to replace the values in the expected value table with current values retrieved from the Parameter Server. A perform state check argument causes the FtTlSCStateCheck to compare the values in the expected value table with current values retrieved from the Parameter Server. The compare is done by calling FoTlExpectedState with the retrieved values.

FtTlStateCheckProxy is a proxy between the StateCheck process and FUI. It relays the command from FUI to the StateCheck process.

FoTlExpectedValue is a class that holds the expected value and high & low values obtained from CMS.

FoTlExpectedState is a class that contains the expected value table. It also performs the compare and replacement of expected values with current values.

FoPsParameters is a class that gets current parameters and stores them in the FoPsClientBuffer.

FoPsClientIF is an interface between the StateCheck and the Parameter Server.

FoPsClientBuffer is a buffer that holds the requested parameters that were requested from the parameter server.

FmMsExpectedStateTable is a proxy between CMS and the Statecheck process. It provides to FtTlSCStateCheck an expected value table which it (FtTlSCStateCheck) requested.

FdEvEventLogger is a class that is a proxy between the Statecheck process and DMS. It receives the event messages generated during the statecheck and relays them to their appropriate destinations.

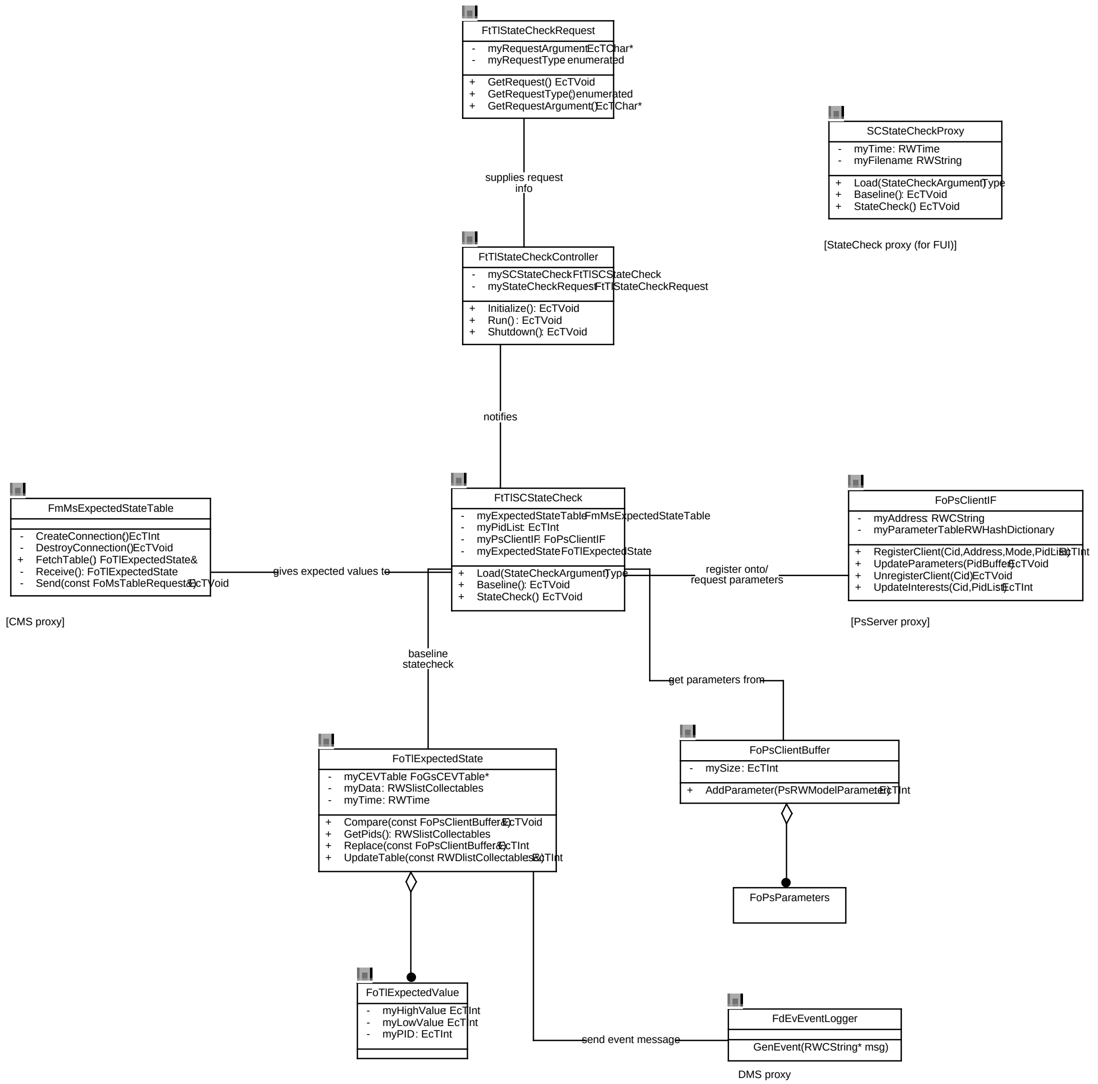

Figure 3.4-2. Spacecraft State Check Object Model

3.4.4 Spacecraft State Check Dynamic Model

The following scenarios are described in this section:

Load Expected State

Initialize State Check

Baseline Expected State

Perform State Check

3.4.4.1 Initialize Spacecraft State Check Scenario

3.4.4.1.1 Initialize Spacecraft State Check Scenario Abstract

This scenario occurs when the StateCheck process is started. It addresses initialization of the StateCheck interfaces and synchronization. The synchronization is achieved by making sure that at least one full master cycle has been decommutated before initialization is complete. The initialize state check event trace is shown in Figure 3.4-3.

3.4.4.1.2Initialize Spacecraft State Check Summary Information

Interfaces:

NameServer

Parameter Server

CMS

FUI

Stimulus:

StateCheck process is started.

Desired Response:

StateCheck process is ready to receive StateCheck commands.

Pre-Conditions:

None.

Post-Conditions:

StateCheck process is ready to receive a load expected value table request.

3.4.4.1.3 Initialize Spacecraft State Check Scenario Description

When a state check process is started, FtTlStateCheckController will initialize its interfaces to Parameter Server and FUI. FtTlStateCheckController then calls the NameServer to get the address of the CMS process and initializes its interface to CMS. FtTlStateCheckController then registers a continuous request for the MasterCycleComplete parameter with FoPsClientIF. FoPsClientIF will return to FtTlStateCheckController who will in turn call FoPsClientIF to get the requested parameter. When the requested parameter is returned to FtTlStateCheckController, and its value indicates that the EOC has received a MasterCycle, the request is canceled by calling FoPsClientIF.

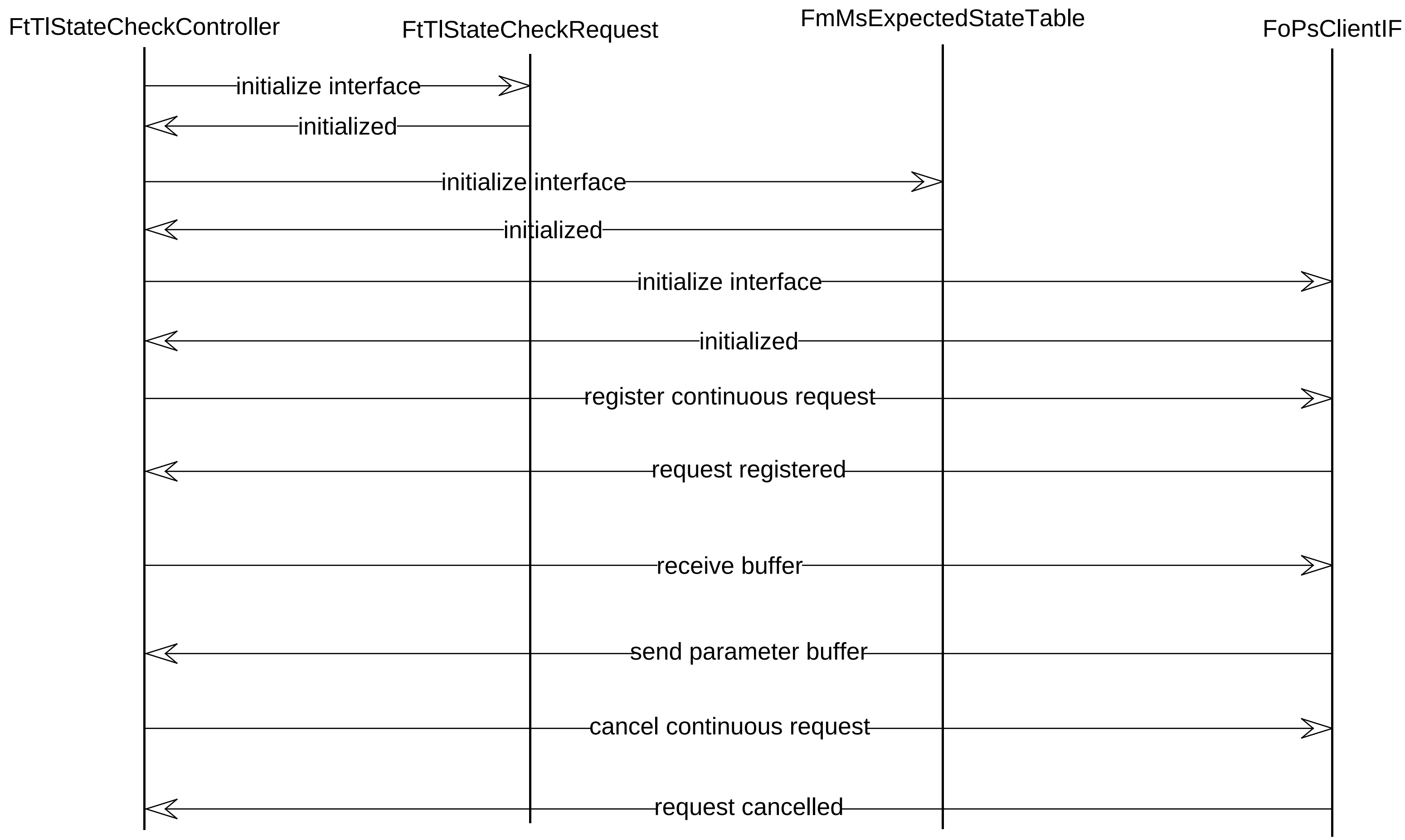

Figure 3.4-3. Initialize Spacecraft State Check Event Trace

3.4.4.2 Load Expected State Table Scenario

3.4.4.2.1 Load Expected State Table Scenario Abstract

This scenario can occur once the StateCheck initialization is complete. This will get a state check load request from FUI and pass it onto CMS. CMS will return the expected value table that can be used at a later time. The load expected state table event trace is shown in Figure 3.4-4.

3.4.4.2.2 Load Expected State Table Summary Information

Interfaces:

FUI

CMS

Stimulus:

StateCheck process receives a load request from FUI.

Desired Response:

StateCheck process loads an expected state table.

Pre-Conditions:

StateCheck has completed initialization.

Post-Conditions:

StateCheck process is ready to receive StateCheck commands.

3.4.4.2.3 Load Expected State Table Scenario Description

FtTlStateCheckController detects that a state check request is available and calls FtTlStateCheck-Request to get the request. FtTlStateCheckController then calls FtTlStateCheckRequest to get the request type. When the type is to load the expected value table, FtTlStateCheckController calls FtTlStateCheckRequest to get the argument. FtTlStateCheckController then calls FtTlSCState-Check with the argument. FtTlSCStateCheck calls FmMsExpectedStateTable with the argument in order to fetch the expected value table. Once CMS provides the expected state table, FtTlSC-StateCheck will loop through each entry in the FoTlExpectedState table in order to accumulate all of the Pids in a single list. This Pid list is used when registering with the parameter server as a one shot client. FtTlSCStateCheck then returns to FtTlStateCheckController.

3.4.4.3 Baseline Expected State Table Scenario

3.4.4.3.1 Baseline Expected State Table Scenario Abstract

This scenario can occur once the StateCheck initialization is complete and an expected state table has been loaded. When a state check baseline request is received from FUI, the values in the expected value table will be replaced by the current values retrieved from the parameter server. The baseline expected state table event trace is shown in Figure 3.4-5.

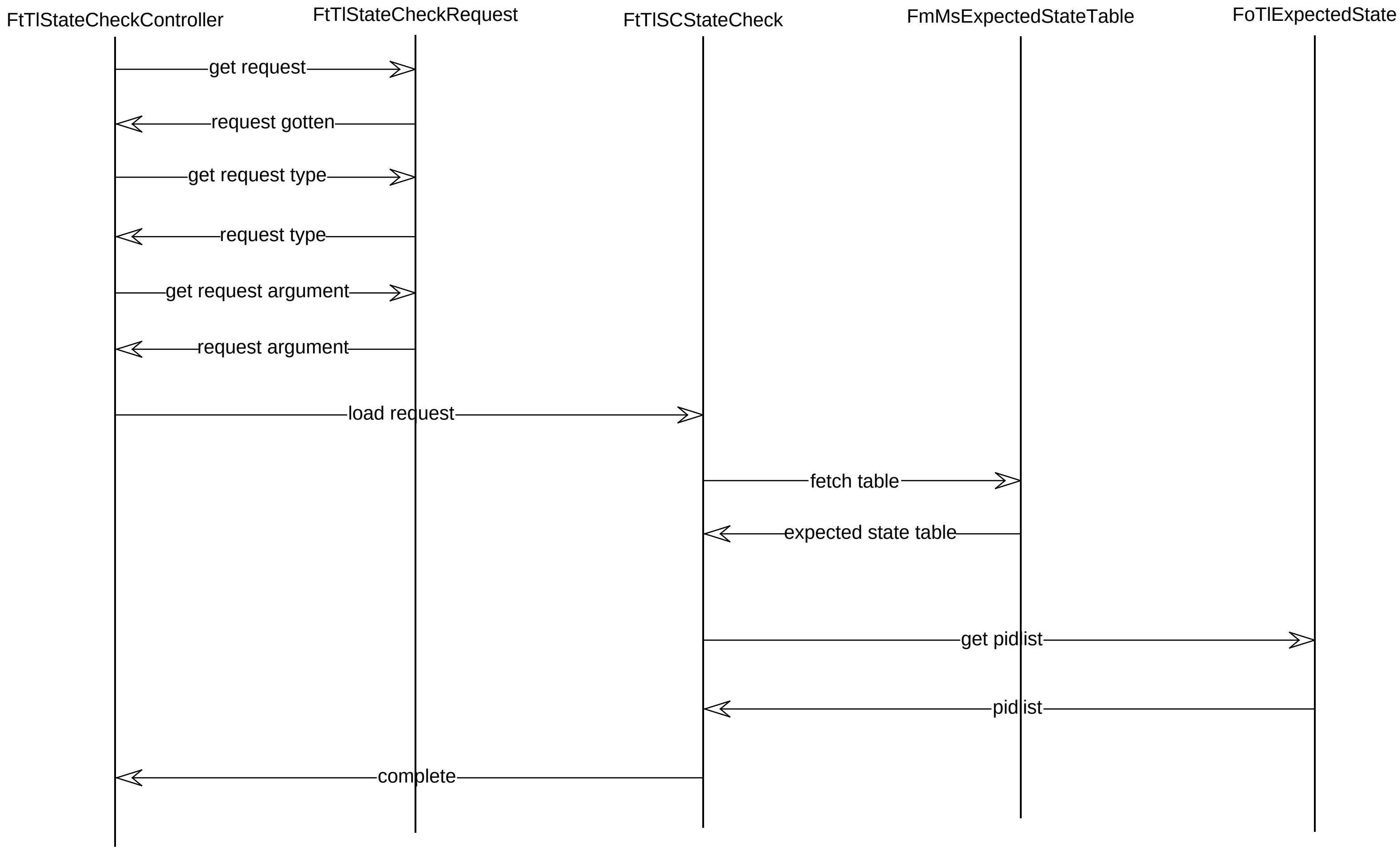

Figure 3.4-4. Load Expected State Table Event Trace

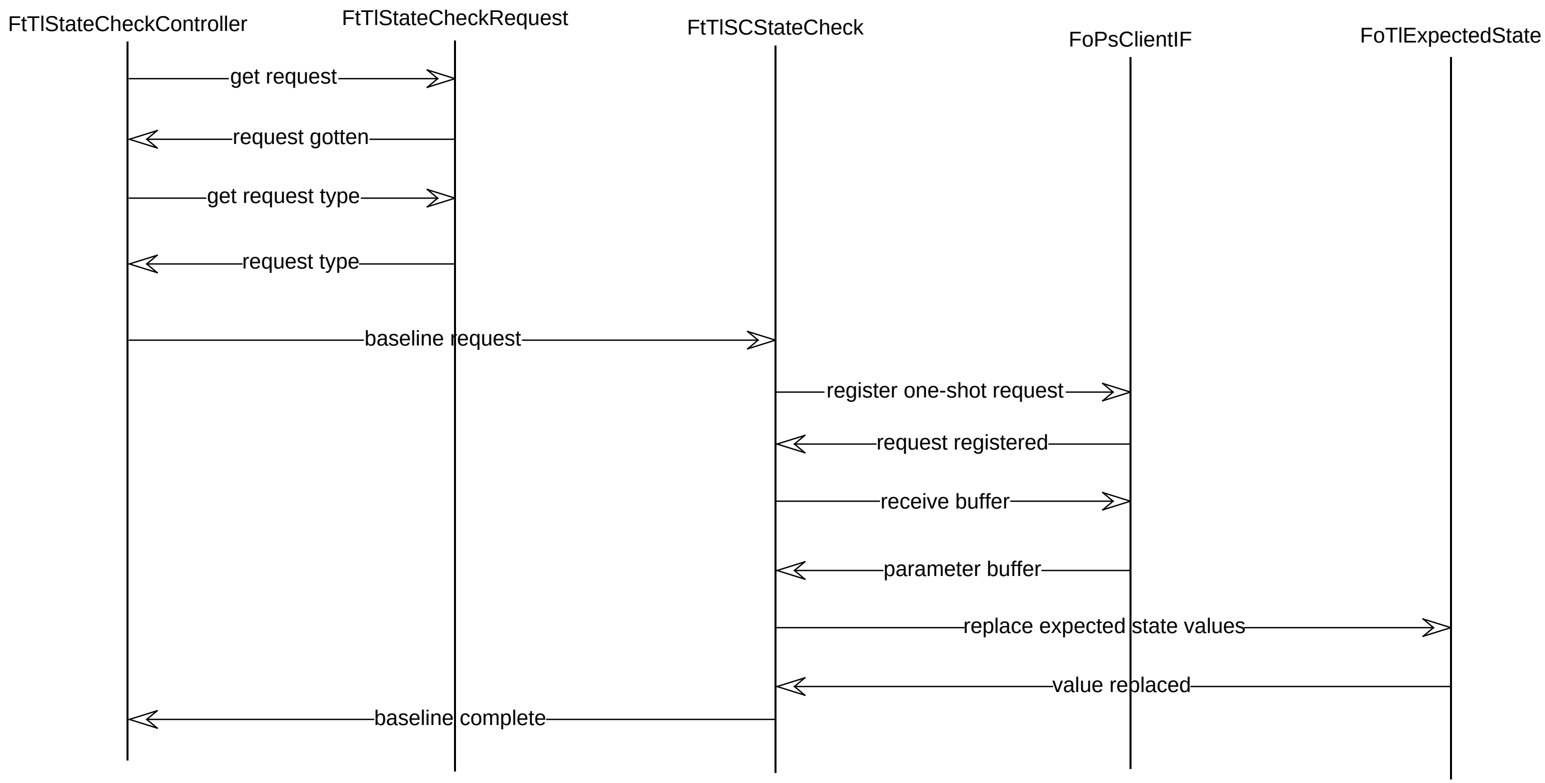

Figure 3.4-5. State Check BaseLine Event Trace

3.4.4.3.2 Baseline Expected State Table Summary Information

Interfaces:

FUI

Parameter Server

Stimulus:

StateCheck process receives a baseline request from FUI.

Desired Response:

StateCheck replaces all values in the expected value table with the current values retrieved from the parameter server.

Pre-Conditions:

StateCheck has completed initialization and an expected state table has been loaded.

Post-Conditions:

StateCheck process is ready to receive StateCheck commands.

3.4.4.3.3 Baseline Expected State Table Scenario Description

FtTlStateCheckController detects that a state check request is available and calls FtTlStateCheck-Request to get the request. FtTlStateCheckController then calls FtTlStateCheckRequest to get the request type. When the type is to baseline the expected value table, FtTlStateCheckController then calls FtTlSCStateCheck to perform the baseline. FtTlSCStateCheck calls FoPsClientIF to register a one-shot request using the previously created Pid list. FoPsClientIF will return to FtTlSCState-Check who will in turn call FoPsClientIF to get the requested parameters. When the requested parameters are returned, FtTlSCStateCheck calls FoTlExpectedState in order to replace the expected state values with the ones retrieved from the parameter server. When the values are replaced and FoTlExpectedState returns, then FtTlSCStateCheck returns to FtTlStateCheckController.

3.4.4.4 Perform Spacecraft State Check Scenario

3.4.4.4.1 Perform Spacecraft State Check Scenario Abstract

This scenario can occur once the StateCheck initialization is complete and an expected state table has been loaded. When a state check request is received from FUI, the values in the expected value table will be compared with the current values retrieved from the parameter server. An event message will be generated for each miscompare, and a summary event message will be generated at the end of a StateCheck. The perform state check event trace is shown in Figure 3.4-6.

3.4.4.4.2 Perform Spacecraft State Check Summary Information

Interfaces:

FUI

Parameter Server

Stimulus:

StateCheck process receives a perform state check request from FUI.

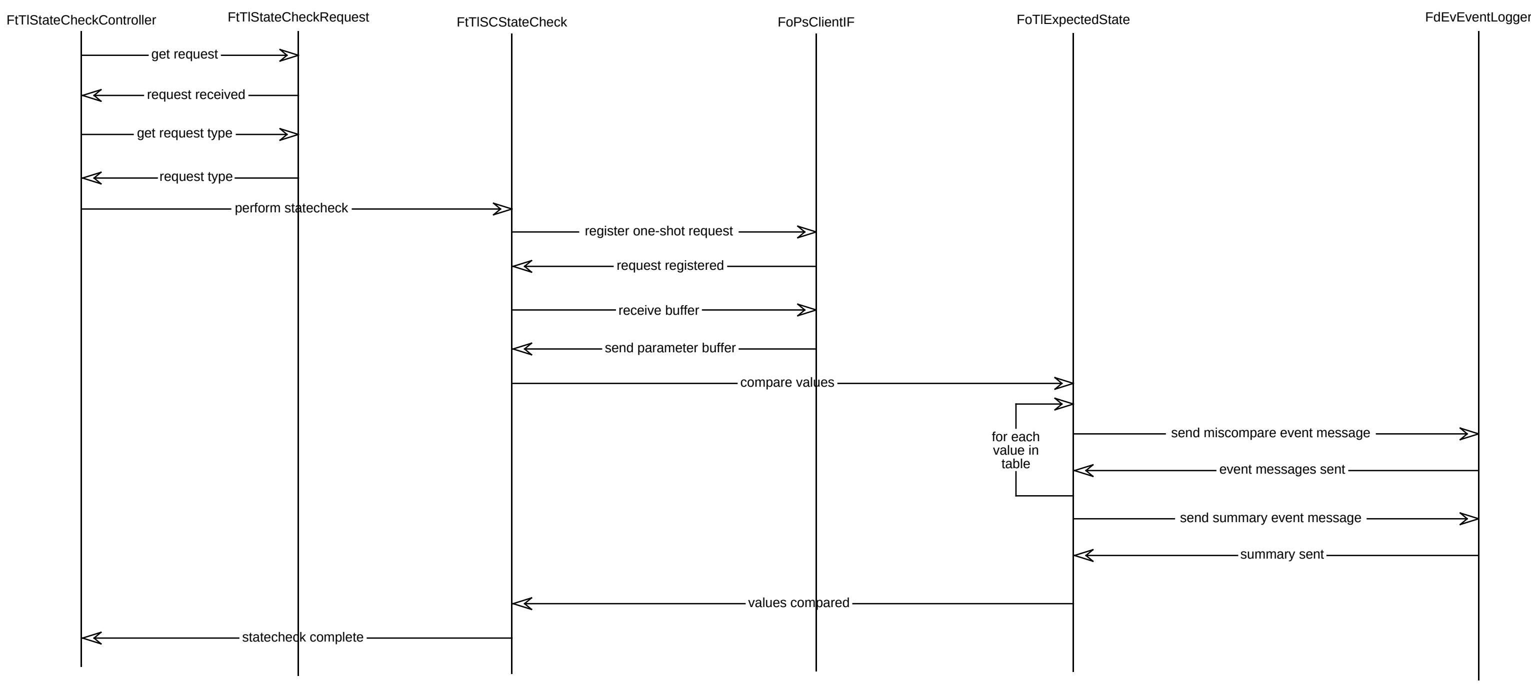

Figure 3.4-6. State Check Perform Event Trace

Desired Response:

StateCheck compares all values in the expected value table with the current values retrieved from the parameter server.

Pre-Conditions:

StateCheck has completed initialization and an expected state table has been loaded.

Post-Conditions:

StateCheck process is ready to receive StateCheck commands.

3.4.4.4.3 Perform Spacecraft State Check Scenario Description

FtTlStateCheckController detects that a state check request is available and calls FtTlStateCheck-Request to get the request. FtTlStateCheckController then calls FtTlStateCheckRequest to get the request type. When the type is to perform the state check, FtTlStateCheckController then calls FtTlSCStateCheck to perform the state check. FtTlSCStateCheck calls FoPsClientIF to register a one-shot request using the previously created Pid list. FoPsClientIF will return to FtTlSCState-Check who will in turn call FoPsClientIF to get the requested parameters. When the requested parameters are returned, FtTlSCStateCheck calls FoTlExpectedState in order to compare the expected state values with the ones retrieved from the parameter server. When the values are compared and FoTlExpectedState returns, then FtTlSCStateCheck returns to FtTlStateCheckController.

3.4.5 Spacecraft State Check Data Dictionary

FdEvEventLogger - class that acts as an interface to DMS.

GenEvent - attribute that generates an event when called.

FmMsExpectedStateTable - class that acts as a proxy to CMS. It supplies a table of expected state values.

FetchTable - attribute that gets a table of values from the database.

FoTlExpectedState - class that provides the list of Pids used to request the parameters from the parameter server.

GetPid - attribute that gets the Pid's.

Compare - attribute that compares the current downlink telemetry value with a range of expected values.

Replace - attribute that replaces the values in the expected value table with current downlink telemetry.

UpdateTable - attribute that updates the table with expected values.

FoTlExpectedValue - class that gets the Pids and high & low values from the database.

FoPsClientBuffer - class that holds the current downlink telemetry values in a buffer.

AddParameter - attribute that adds parameters into the buffer.

FoPsClientIF - class that serves as a proxy between the parameter server and Statecheck process.

RegisterClient - attribute that registers clients onto the parameter server.

UpdateParameters - attribute that updates parameter fields with current parameters.

UnregisterClient - attribute that unregisters clients off the parameter server.

UpdateInterests - attribute that updates interested clients .

FoPsParameters - class that contains the parameters.

FtTlSCStateCheck - class that assists in back orbit verification. It can monitor and compare the spacecraft's state with an expected state.

Load - attribute that loads the expected values into an expected value table.

Baseline - operation that replaces the expected value table with current downlink telemetry values.

StateCheck - operation that compares the downlink telemetry values with the expected state.

FtTlStateCheckController - class that is responsible for controlling an instance of the state check subsystem process. This class receives and processes configuration adjustment requests.

Initialize - operation that initializes attributes and interfaces.

Run - operation that runs the state check controller process.

Shutdown - operation that shuts down the state check controller process.

FtTlStateCheckRequest - class that acts as a proxy. It receives requests and relays the appropriate procedure to FtTlStateCheckController.

GetRequestType - attribute that returns request type.

GetTable - attribute that gets the table of Pid's and Values.

3.5 Parameter Server

The Parameter Server process is responsible for providing a central repository where processes can go to receive continuous or one shot parameter updates. Processes can also go to the Parameter Server to update parameters that they generate and are responsible for providing to other processes. Every process that updates parameters must initialize any parameter values that they serve to a default value so that anyone that requests a parameter can be assured of receiving some value in return.

Requests for service are handled via the Parameter Server's interface. A proxy interface object is provided to any process that requires the services of the Parameter Server. Clients make requests via calls to the interface object, thus hiding much of the interprocess communication and data formatting/object flattening mechanisms.

3.5.1 Parameter Server Context

The Parameter Server process receives two types of requests, one type to serve out parameters to clients and another type of request to update parameters that are held in the Parameter Server. Any client can use these services provided it knows what it wants or what it wants to update. The Parameter Server context diagram is shown in figure 3.5-1.

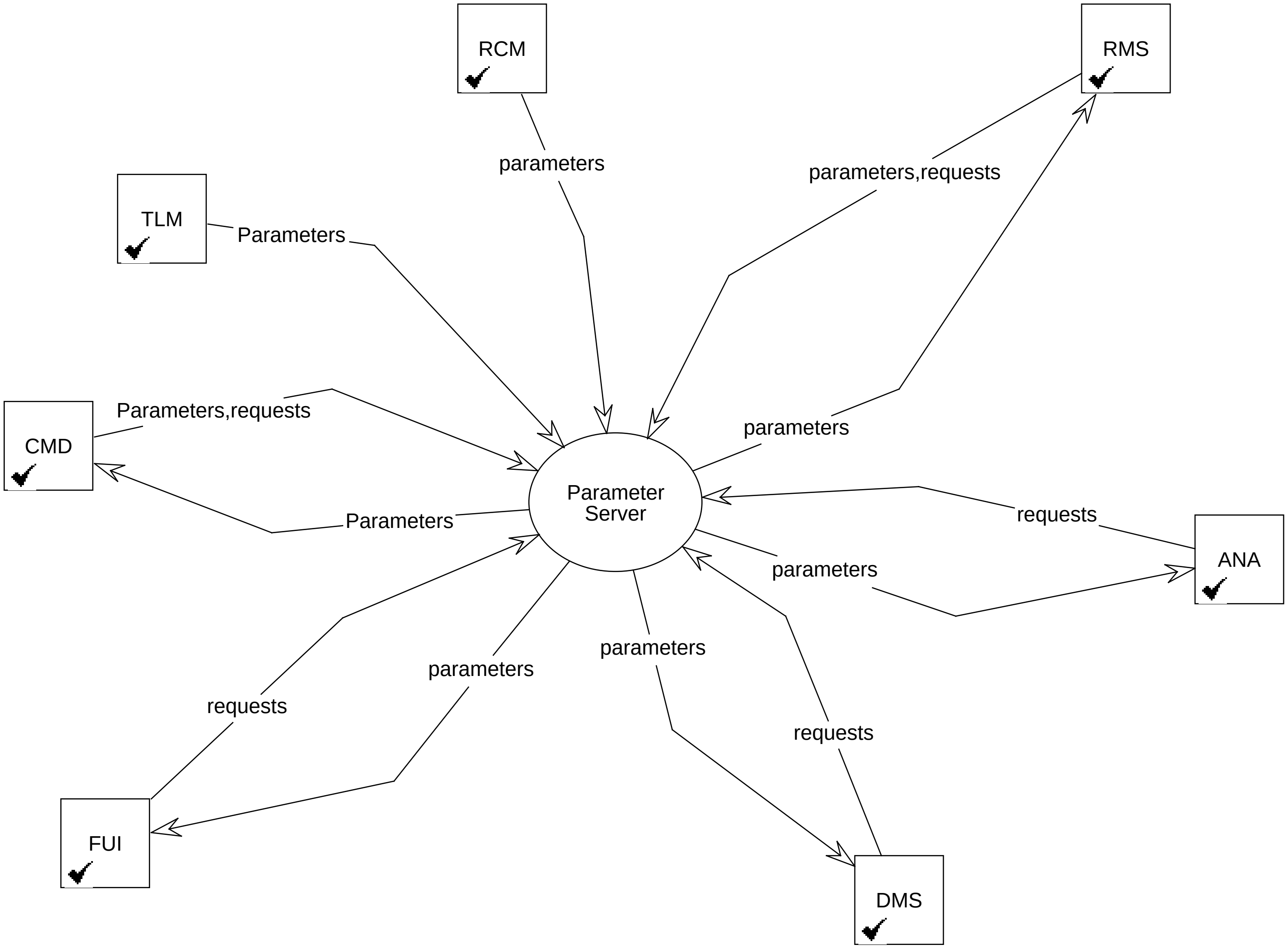

Figure 3.5-1. Parameter Server Context Diagram

3.5.2 Parameter Server Interfaces

Table 3.5-1. Parameter Server Interfaces

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Register Client	FoPsClientIF	Register a client as continuous or oneshot	Param Server	RMS, FUI, CMD, ANA, DMS	on process startup and as one shot clients are needed
Update Parameters	FoPsClientIF	Allow providers to update parameter server	Param Server	RCM, RMS, TLM, CMD	TLM every packet; others as needed
Unregister Client	FoPsClientIF	Allow client to disconnect from parameter server	Param Server	RMS, FUI, CMD, ANA, DMS	on process shutdown
Update Interests	FoPsClientIF	Allow client to change its interests	Param Server	RMS, FUI, CMD, ANA, DMS	rare
AddParam to-Buffer	FoPsClientIF	Allow client to build buffer of parameters to update	Param Server	RCM, RMS, TLM, CMD	whenever a parameter needs to be updated
Receive Buff-er	FoPsClientIF	Allow client to receive param-eter buffer	Param Server	RMS, FUI, CMD, ANA, DMS	whenever data is sent to clients

3.5.3 Parameter Server Object Model

The Parameter Server is both a repository and a mechanism for sharing data between processes. It is designed such that it can live on its own or inside another process. The object model for the parameter server (Figure 3.5-2) was developed to support such an idea. The parameter server is designed to accept service calls for two types of services. One service is providing parameters to any process that has a need for them. The other service is allowing the parameter providers the ability to update the parameters as new values are obtained.

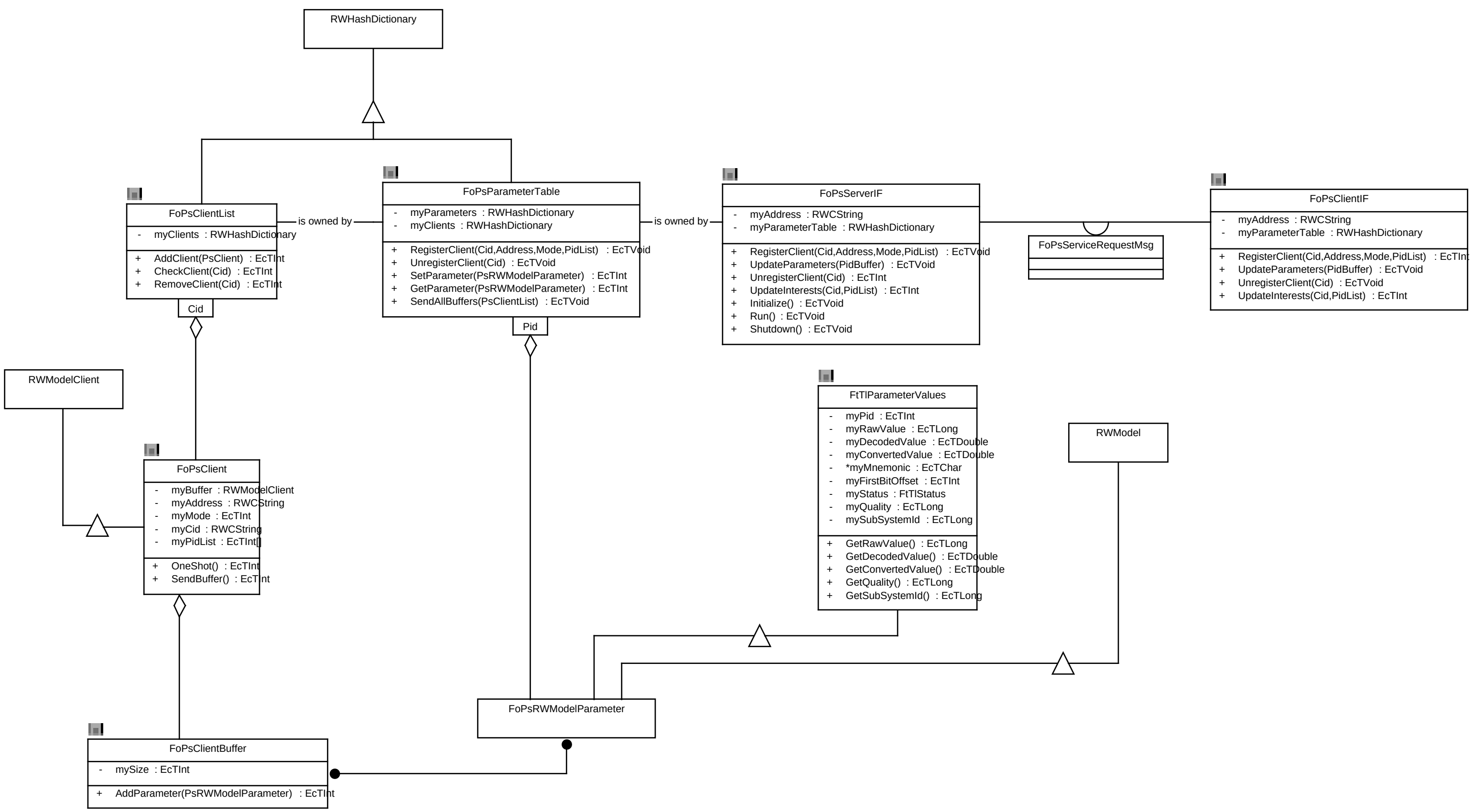

Figure 3.5-2. Parameter Server Object Model

Requests for service are handled in the parameter server interface. The FoPsClientIF object is given to any process that needs services from the parameter server. When the client makes a service call of the FoPsClientIF object it is transmitted to the FoPsServerIF via interprocess communications. Once the request is received in the FoPsServerIF object it is translated back to a service call of the FoPsParameterTable. The FoPsParameterTable object is the repository for all of the parameters that it collects. A parameter ID is the key to obtaining a parameter from the FoPsParameterTable. All updates to the parameter objects are made through the FoPsParameterTable as well as requests for parameters themselves. The FoPsParameterTable object also keeps a list of clients and their respective process information. That is held in the FoPsClientList object. That list contains FoPsClient objects which represent the external clients to the Parameter Server.

FoPsClientIF is the clients view of the parameter server. A client uses this interface to request services from the parameter server.

FoPsServerIF is the servers view of the parameter server. The parameter server uses this object to catch all incoming requests of the parameter server.

FoPsParameterTable is a container class of FoPsRWModelParameters. It also performs the request processing of the parameter server.

FoPsClientList is a container class of FoPsClient objects. It holds all of the client objects.

FoPsClient is an object that represents the external client of the parameter server. It contains information on client requests and the clients address and process id.

FoPsClientBuffer is an object that holds the parameters that get served to an individual client.

FoPsServiceRequestMsg is an object that represents the client information passed from the proxy to the server.

3.5.4 Parameter Server Dynamic Model

The following are scenarios defined in this section.

Register a client as a continuous user

Register a client as a one shot user

Send buffer to continuous client

Update the interests of a continuous client

Update parameters from parameter provider

3.5.4.1 Register a Client as a Continuous User Scenario

3.5.4.1.1 Register a Client as a Continuous User Scenario Abstract

The purpose of this scenario is to describe the process by which a client is registered for continuous parameter serving. Figure 3.5-3 is the event trace for this scenario.

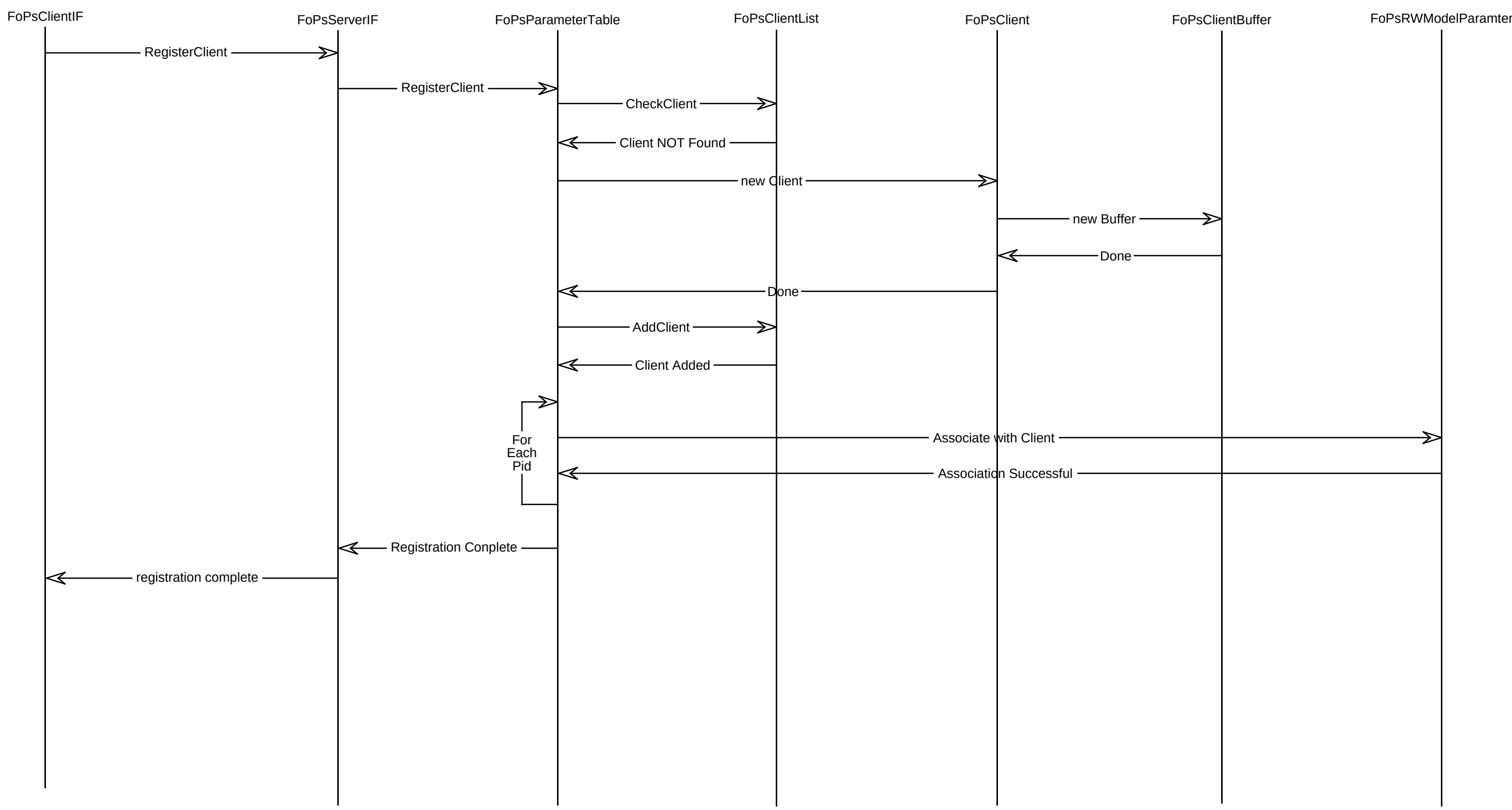

Figure 3.5-3. Register a Continuous User Event Trace

3.5.4.1.2 Register a Client as a Continuous User Summary Information

Interfaces:

An external client

Stimulus:

A client request for continuous parameter updates

Desired Response:

The client receives continuous updates of the parameters it requested

Pre-Conditions:

The parameter server is ready to accept incoming requests

Post-Conditions:

The parameter server is ready to accept incoming requests

3.5.4.1.3 Register a Client as a Continuous User Scenario Description

The client calls the RegisterClient operation of the FoPsClientIF. The FoPsClientIF, in turn, calls the RegisterClient operation of the FoPsServerIF object through interprocess object passing. Once the call is made to the FoPsServerIF it calls the RegisterClient service of the

FoPsParameterTable. The FoPsParameterTable will determine that the mode of this registration is continuous and create a new FoPsClient and add it to the FoPsClientList. Then it will associate the FoPsClient with the parameters that it is interested by way of the parameter ID list that the FoPsClient keeps. Once the associations are made the status of this call is returned back through all of the called objects back to the FoPsClientIF and back to the client.

3.5.4.2 Register a Client as a One Shot User Scenario

3.5.4.2.1 Register a Client as a One Shot User Scenario Abstract

The purpose of this scenario is to describe the process by which a client is registered for one shot parameter serving. Figure 3.5-4 is the event trace for this scenario.

3.5.4.2.2 Register a Client as a One Shot User Summary Information

Interfaces:

An external client

Stimulus:

A client request for a one shot parameter update

Desired Response:

The client receives a one time update of the parameters it requested

Pre-Conditions:

The parameter server is ready to accept incoming requests

Post-Conditions:

The parameter server is ready to accept incoming requests

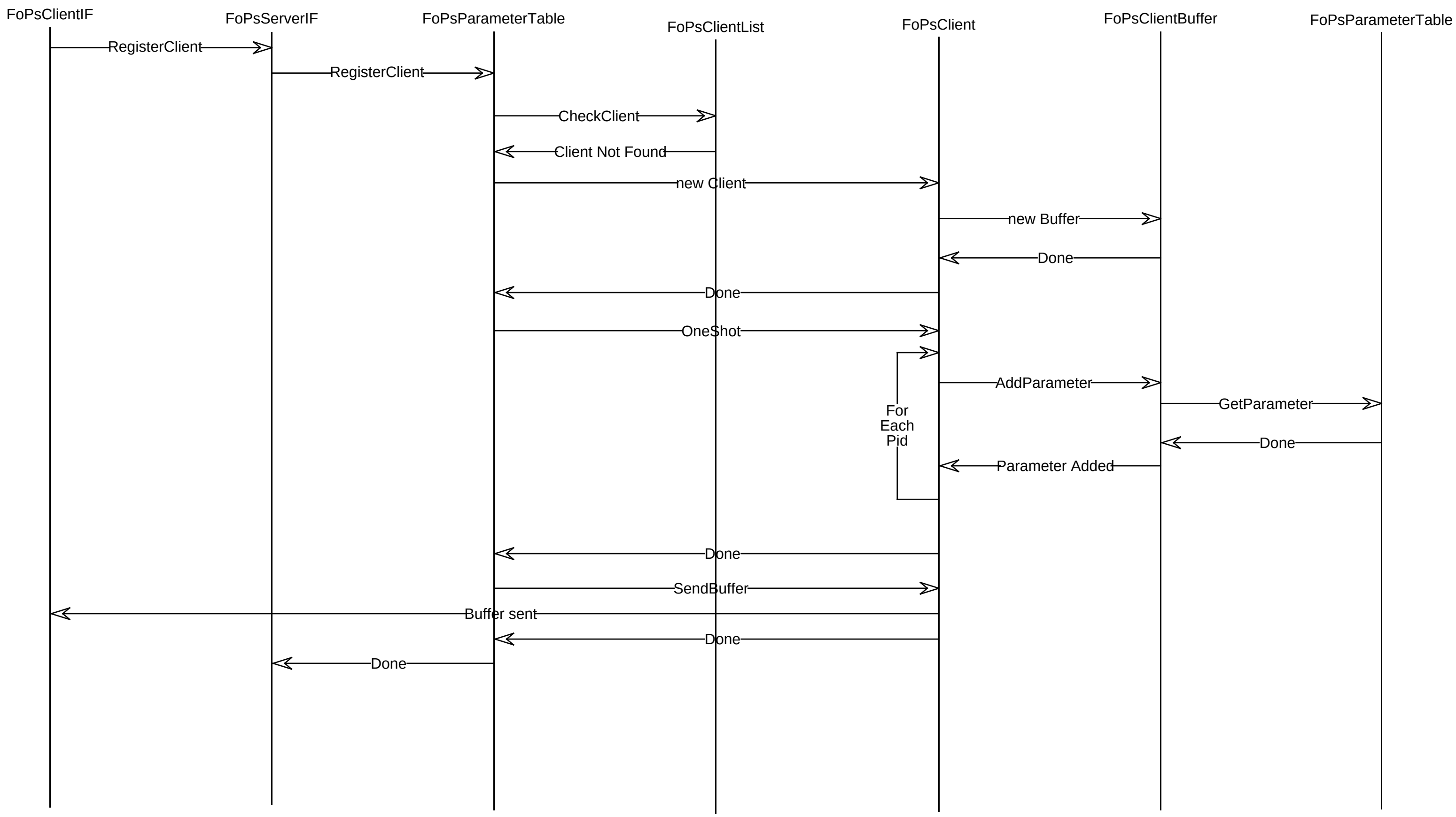

Figure 3.5-4. Register a One Shot User Event Trace

3.5.4.2.3 Register a Client as a One Shot User Scenario Description

The client calls the RegisterClient operation of the FoPsClientIF. The FoPsClientIF, in turn, calls the RegisterClient operation of the FoPsServerIF object through interprocess object passing. Once the call is made to the FoPsServerIF it calls the RegisterClient service of the FoPsParameterTable. The FoPsParameterTable will determine that the mode of this registration is one shot and create a new FoPsClient. It will then cycle through all of the parameter IDs in the client's parameter ID list and get each parameter from the FoPsParameterTable and place it in the FoPsClientBuffer. Once all of the parameters are in the buffer it is sent back to the client through the FoPsClientIF.

3.5.4.3 Send Buffer to a Continuous Client Scenario

3.5.4.3.1 Send Buffer to a Continuous Client Scenario Abstract

The purpose of this scenario is to describe the process by which a client is registered for continuous parameter serving. Figure 3.5-5 is the event trace for this scenario.

3.5.4.3.2 Send Buffer to a Continuous Client Summary Information

Interfaces:

An external client

Stimulus:

A client request for continuous parameter updates

Desired Response:

The client receives a buffer of the parameters it requested

Pre-Conditions:

The parameter server has received updates of parameters requested by the client

Post-Conditions:

The parameter server is ready to accept incoming requests

3.5.4.3.3 Send Buffer to a Continuous Client Scenario Description

The Parameter Server will make updates to all of the parameters from a provider and determine if any clients are interested in them. This client has an interest in a parameter that was updated so it is put in the clients buffer. When the provider is done updating the parameters the FoPsParameterTable will call its SendAllBuffers operation which will in turn call the SendBuffer operation of all of the FoPsClients that have buffers that have at least one parameter in them and they will be sent to each client.

3.5.4.4 Update the Interests of a Client Scenario

3.5.4.4.1 Update the Interests of a Client Scenario Abstract

The purpose of this scenario is to describe the process by which a client updates the parameters in which it has an interest in receiving. Figure 3.5-6 is the event trace for this scenario.

3.5.4.4.2 Update the Interests of a Client Summary Information

Interfaces:

An external client

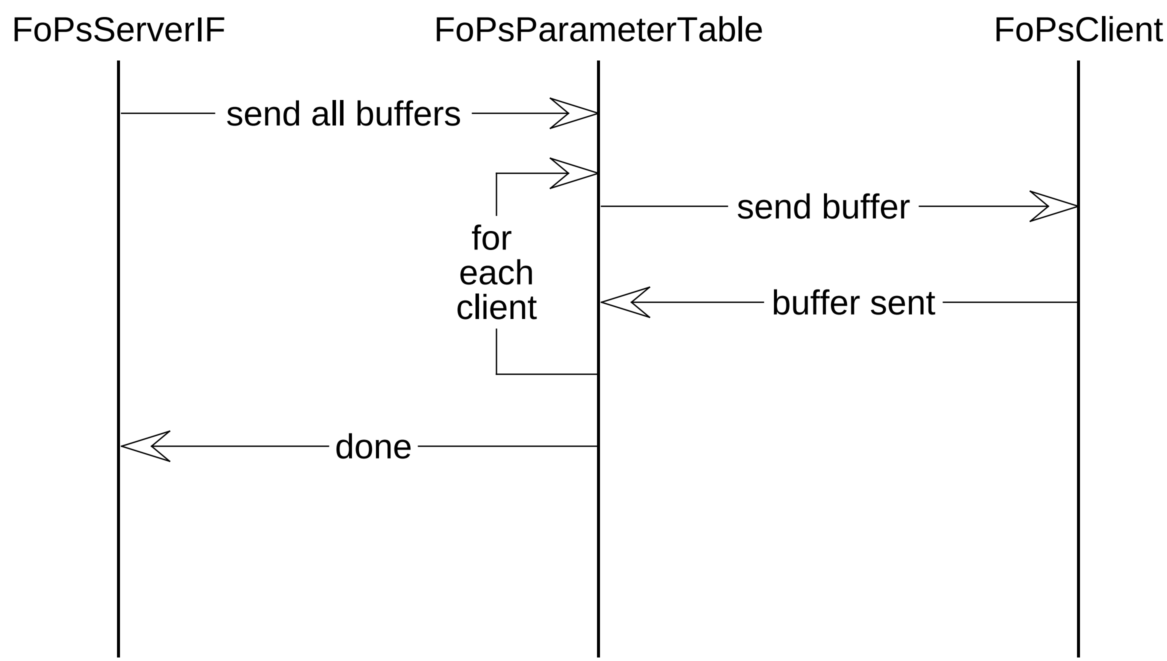

Figure 3.5-5. Send Buffer to Continuous Client Event Trace

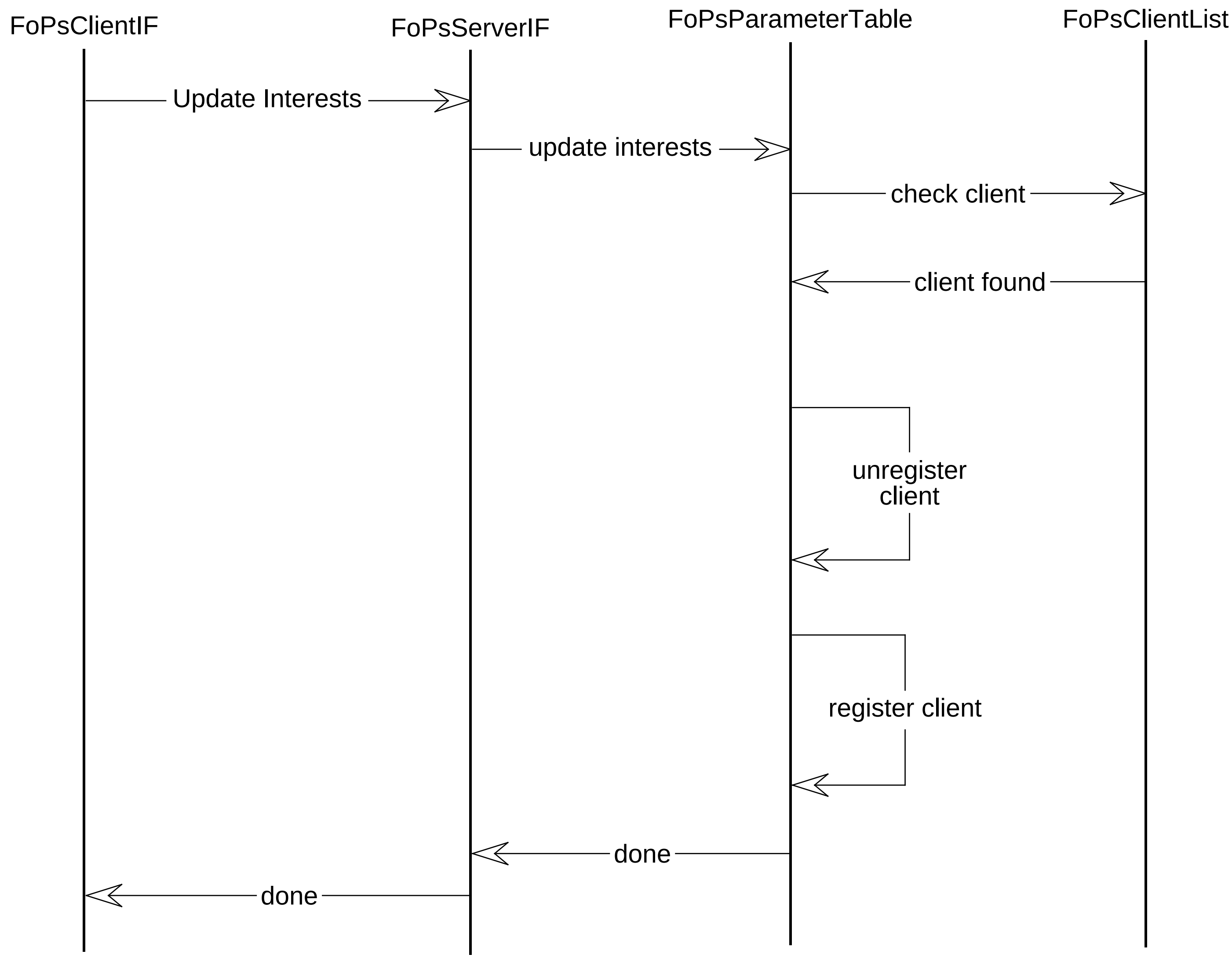

Figure 3.5-6. Update Client Interests Event Trace

Stimulus:

A client requests an update of his interests

Desired Response:

The client information of his interests is updated

Pre-Conditions:

The parameter server is ready to accept incoming requests

Post-Conditions:

The parameter server is ready to accept incoming requests

3.5.4.4.3 Update the Interests of a Client Scenario Description

The client will call the UpdateInterests operation of the FoPsClientIF who will in turn call the UpdateInterests operation of the FoPsServerIF. Once in the FoPsServerIF it will call the UpdateInterests operation of the FoPsParameterTable. It will check to see if the client exists or not and if the client exists it will call its UnregisterClient operation to remove the old parameter associations and then it will call its RegisterClient operation to essentially re-register the client with its new interests and then return the status back through the called objects to the FoPsClientIF which will relay the status of the call to the client.

3.5.4.5 Update Parameters from a Parameter Provider Scenario

3.5.4.5.1 Update Parameters from a Parameter Provider Scenario Abstract

The purpose of this scenario is to describe the process by which a parameter producer updates parameters in the parameter server. Figure 3.5-7 is the event trace for this scenario.

3.5.4.5.2 Update Parameters from a Parameter Provider Summary Information

Interfaces:

An external client

Stimulus:

A client requests to update parameters

Desired Response:

The parameters the client provides are updated in the parameter table

Pre-Conditions:

The parameter server is ready to accept incoming requests

Post-Conditions:

The parameter server is ready to accept incoming requests

3.5.4.5.3 Update Parameters from a Parameter Provider Scenario Description

For each parameter that the client wants to update it will call the AddParamToBuffer operation on the FoPsClientIF. That will put the parameter in a temporary buffer. Once all of the parameters are in the temporary buffer the client will call the UpdateParameters operation of the FoPsClientIF. The FoPsClientIF will call the UpdateParameters operation of the FoPsServerIF who will in turn call the UpdateParameters operation of the FoPsParameterTable. The FoPsClientIF will then return to the control to the client so that the client can go back and do whatever it needs to do without waiting for the parameters to update. The FoPsParameterTable will take each parameter from the temporary buffer and update the corresponding parameter in the table with the new information.

3.5.5 Parameter Server Data Dictionary

FoPsClient

class **FoPsClient**

This class is used to create, destroy and process any client which expresses an interest in any parameter contained within the parameter table.

Public Construction

FoPsClient(RWCString myAddress, EcTInt myMode, RWCString myCid, EcTInt myPidList)

This member function is the default constructor for this class.

~FoPsClient(EcTVoid)

FoPsClient

This member function is the destructor for this class.

Public Functions

EcTInt **OneShot**(EcTVoid)

This member function will process a client which is requesting that Pid values be sent to him once only. Hence the name OneShot.

EcTInt **SendBuffer**(EcTVoid)

This member function will send the buffer which contains the requested Pid's to its respective client.

Private Data

RWCString **myAddress**

This member variable identifies the address/port of the client

RWModelClient **myBuffer**

This member variable identifies the address of the clients Pid buffer

RWCString **myCid**

This member variable identifies the client by id/name

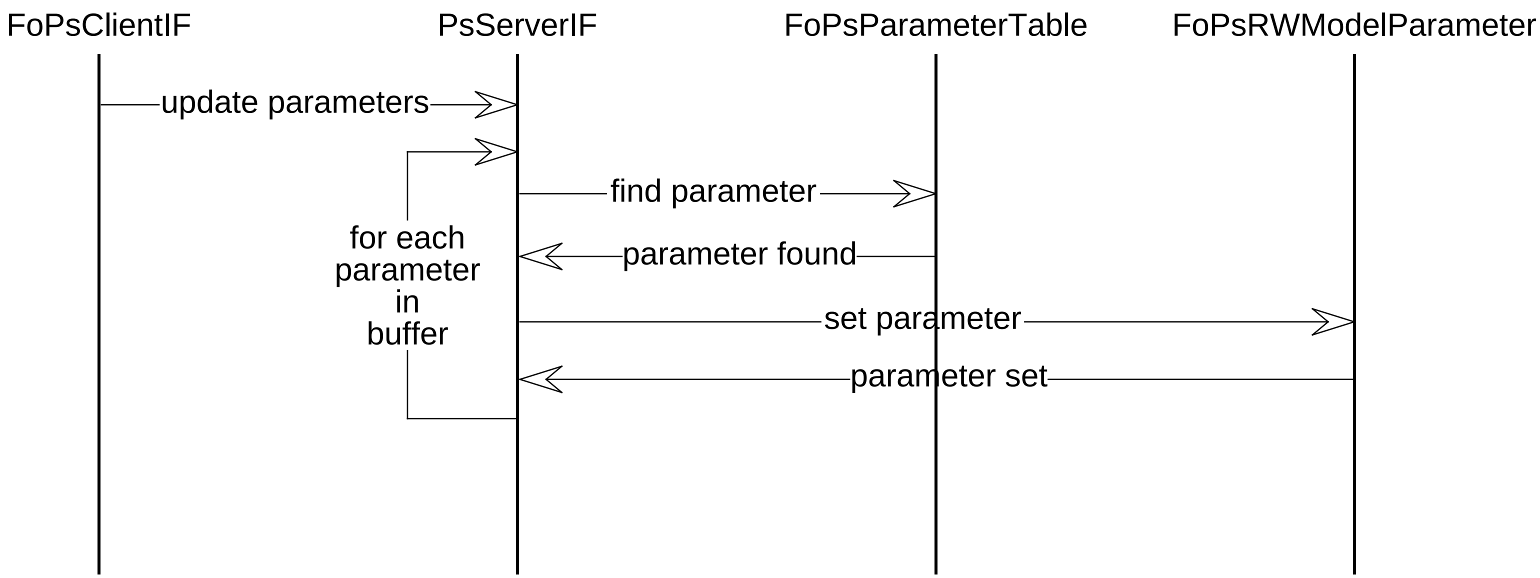

Figure 3.5-7. Update Parameters from Provider Event Trace

EcTInt **myMode**

This member variable identifies the type/mode of client making the request, i.e. "continuous" or "one-shot".

EcTInt **myPidList**[]

This member variable contains the Pid's which the requesting client has an interest.

FoPsClientBuffer

class **FoPsClientBuffer**

This class is used to create and destroy a client buffer along with the capability of adding parameters to the clients parameter buffer.

Public Construction

FoPsClientBuffer(EcTVoid)

This member function is the default constructor for this class.

~FoPsClientBuffer(EcTVoid)

This member function is the destructor for this class

Public Functions

EcTInt **AddParameter**(FoPsRWModelParameter)

This member function is used for adding parameters to the clients parameter buffer.

Private Data

EcTInt **mySize**

This member variable identifies the current buffer size

FoPsClientIF

class **FoPsClientIF**

This class is the client representation of a parameter server. This object is given to all processes that which to use the services of a parameter server.

Base Classes

public **PsParameterServerIF**

Public Functions

EcTEcTVoid **FoPsServerIF**()

This member function is the default constructor for this class.

EcTInt **RegisterClient**(RWCString Cid, RWCString Address, EcTInt Mode, EcTInt PidList[])

This member function allows the client to register to receive either a oneshot request or a continuous request for parameters.

EcTEcTVoid **UnregisterClient**(RWCString Cid)

This member function allows a client to unregister an interest in parameters.

EcTInt **UpdateInterests**(RWCString Cid, EcTInt PidList[])

This member function allows a client to change his parameter interests.

EcTEcTVoid **UpdateParameters**(FoPsClientBuffer PidBuffer)

This member function allows a parameter producer to update the parameters that it generates.

Private Data

RWCString **myAddress**

This member variable is the address of the parameter server.

FoPsClientList

class **FoPsClientList**

This class is used to contain client objects who have contacted the parameter server with a registered interest of specific parameters.

Public Construction

FoPsClientList()

This member function is the default constructor for this class.

~FoPsClientList()

This member function is the destructor for this class

Public Functions

EcTInt **AddClient**(FoPsClient)

This member function adds a client to the client list

EcTInt **CheckClient**(Cid)

This member function searches the client list to see if the client already is registered.

EcTInt **RemoveClient**(Cid)

This member function removes a client from the client list

FoPsParameterTable

class **FoPsParameterTable**

This class is the keeper of all of the parameters for a parameter server. Anything that you might want out of a parameter can be gotten through the parameter table.

Public Functions

EcTEcTVoid **FoPsParameterTable**()

This member function is the default constructor for this class.

EcTInt **GetParameter**(FoPsRWModelParameter Param)

This member function will get a parameter and return it to the requester.

EcTEcTVoid **RegisterClient**(RWCString Cid, RWCString Address, EcTInt Mode, EcTInt PidList[])

This member function will register a client with the parameter table.

EcTEcTVoid **SendAllBuffers**(FoPsClientList myClients)

This member function will send all the client buffers to their respective clients.

EcTInt **SetParameter**(FoPsRWModelParameter Param)

This member function will set the values of a parameter.

EcTEcTVoid **UnregisterClient**(RWCString Cid)

This member function will remove a client from the parameter server.

Private Data

FoPsClientList **myClients**

This member variable is the object that holds all of the clients.

FoPsServerIF

class **FoPsServerIF**

This class represents the server side of the interface between the parameter server and the outside world.

Base Classes

public **FoPsParameterServerIF**

Public Functions

EcTEcTVoid **FoPsServerIF**()

This member function is the default constructor for this class.

EcTEcTVoid **Initialize**()

This member function initializes all of the objects that are needed to start a parameter server.

EcTEcTVoid **RegisterClient**(RWCString Cid, RWCString Address, EcTInt Mode, EcTInt PidList[])

This member function will register a client to receive continuous parameter updates or a one-shot of parameters.

EcTEcTVoid **Run**()

This member function will start the parameter server and get the parameter server ready to accept requests for parameters and requests to update parameters.

EcTEcTVoid **Shutdown**()

This member function will nicely terminate a parameter server.

EcTInt **UnregisterClient**(RWCString Cid)

This member function allows the client to terminate a request for continuous parameter updates.

EcTInt **UpdateInterests**(RWCString Cid, EcTInt PidList[])

This member function allows a client to modify the list of parameters that it has an interest in.

EcTEcTVoid **UpdateParameters**(PidBuffer)

This member function allows the parameter provider to update parameters that it generates.

Private Data

RWCString **myAddress**

This member variable is the clients address

FoPsParameterTable **myParameterTable**

This member variable is the parameter table.

ServiceRequestMessage

class **ServiceRequestMessage**

Base Classes

public **RWCollectable**

Protected Data

RWOrdered **myServiceArgList**

This member variable corresponds to the argument list to be passed to the parameter server.

eService **myServiceId**

attributes myServiceId

This member variable corresponds to the Id number of the service being requested.

Inherited from class "RWCollectable"

eService

enum **eService**

This member variable will define available services

This page intentionally left blank.

Abbreviations and Acronyms

ACL	Access Control List
AD	Acceptance Check/TC Data
AGS	ASTER Ground System
AM	Morning (ante meridian) -- see EOS AM
Ao	Availability
APID	Application Identifier
ARAM	Automated Reliability/Availability/Maintainability
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer (formerly ITIR)
ATC	Absolute Time Command
BAP	Baseline Activity Profile
BC	Bypass check/Control Commands
BD	Bypass check/TC Data (Expedited Service)
BDU	Bus Data Unit
bps	bits per second
CAC	Command Activity Controller
CCB	Change Control Board
CCSDS	Consultative Committee for Space Data Systems
CCTI	Control Center Technology Interchange
CD-ROM	Compact Disk-Read Only Memory
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CERES	Clouds and Earth's Radiant Energy System
CI	Configuration item
CIL	Critical Items List
CLCW	Command Link Control Words
CLTU	Command Link Transmission Unit
CMD	Command subsystem
CMS	Command Management Subsystem
CODA	Customer Operations Data Accounting
COP	Command Operations Procedure
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit

CRC	Cyclic Redundancy Code
CSCI	Computer software configuration item
CSMS	Communications and Systems Management Segment
CSS	Communications Subsystem (CSMS)
CSTOL	Customer System Test and Operations Language
CTIU	Command and Telemetry Interface Unit (AM-1)
DAAC	Distributed Active Archive Center
DAR	Data Acquisition Request
DAS	Detailed Activity Schedule
DAT	Digital Audio Tape
DB	Data Base
DBA	Database Administrator
DBMS	Database Management System
DCE	Distributed Computing Environment
DCP	Default Configuration Procedure
DEC	Digital Equipment Corporation
DES	Data Encryption Standard
DFCD	Data Format Control Document
DID	Data Item Description
DMS	Data Management Subsystem
DOD	Digital Optical Data
DoD	Department of Defense
DS	Data Server
DSN	Deep Space Network
DSS	Decision Support System
e-mail	electronic mail
Ecom	EOS Communication
ECS	EOSDIS Core System
EDOS	EOS Data and Operations System
EDU	EDOS Data Unit
EGS	EOS Ground System
EOC	Earth Observation Center (Japan); EOS Operations Center (ECS)
EOD	Entering Orbital Day
EON	Entering Orbital Night
EOS	Earth Observing System

EOSDIS	EOS Data and Information System
EPS	Encapsulated Postscript
ESH	EDOS Service Header
ESN	EOSDIS Science Network
ETS	EOS Test System
EU	Engineering Unit
EUVE	Extreme Ultra Violet Explorer
FAS	FOS Analysis Subsystem
FAST	Fast Auroral Snapshot Explorer
FDDI	Fiber Distributed Data Interface
FDF	Flight Dynamics Facility
FDIR	Fault Detection and Isolation Recovery
FDM	FOS Data Management Subsystem
FMEA	Failure Modes and Effects Analyses
FOP	Frame Operations Procedure
FORMATS	FDF Orbital and Mission Aids Transformation System
FOS	Flight Operations Segment
FOT	Flight Operations Team
FOV	Field-Of-View
FPS	Fast Packet Switch
FRM	FOS Resource Management Subsystem
FSE	FOT S/C Evolutions
FTL	FOS Telemetry Subsystem
FUI	FOS User Interface
GB	Gigabytes
GCM	Global Circulation Model
GCMR	Global Circulation Model Request
GIMTACS	GOES I-M Telemetry and Command System
GMT	Greenwich Mean Time
GN	Ground Network
GOES	Geostationary Operational Environmental Satellite
GSFC	Goddard Space Flight Center
GUI	Graphical User Interface
H&S	Health and Safety
H/K	Housekeeking
HST	Hubble Space Telescope

I/F	Interface
I/O	Input/Output
ICC	Instrument Control Center
ICD	Interface Control Document
ID	Identifier
IDB	Instrument Database
IDR	Incremental Design Review
IEEE	Institute of Electrical and Electronics Engineers
IOT	Instrument Operations Team
IP	International Partners
IP-ICC	International Partners-Instrument Control Center
IPs	International Partners
IRD	Interface requirements document
ISDN	Integrated Systems Digital Network
ISOLAN	Isolated Local Area Network
ISR	Input Schedule Request
IST	Instrument Support Terminal
IST	Instrument Support Toolkit
IWG	Investigator Working Group
JPL	Jet Propulsion Laboratory
Kbps	Kilobits per second
LAN	Local Area Network
LaRC	Langley Research Center
LASP	Laboratory for Atmospheric Studies Project
LEO	Low Earth Orbit
LOS	Loss of Signal
LSM	Local System Manager
LTIP	Long-Term Instrument Plan
LTSP	Long-Term Science Plan
MAC	Medium Access Control; Message Authentication Code
MB	Megabytes
MBONE	Multicast Backbone
Mbps	Megabits per second
MDT	Mean Down Time
MIB	Management Information Base

MISR	Multi-angle Imaging Spectro-Radiometer
MMM	Minimum, Maximum, and Mean
MO&DSD	Mission Operations and Data Systems Directorate (GSFC Code 500)
MODIS	Moderate resolution Imaging Spectrometer
MOPITT	Measurements Of Pollution In The Troposphere
MSS	Management Subsystem
MTPE	Mission to Planet Earth
NASA	National Aeronautics and Space Administration
Nascom	NASA Communications Network
NASDA	National Space Development Agency (Japan)
NCAR	National Center for Atmospheric Research
NCC	Network Control Center
NEC	North Equator Crossing
NFS	Network File System
NOAA	National Oceanic and Atmospheric Administration
NSI	NASA Science Internet
NTT	Nippon Telephone and Telegraph
OASIS	Operations and Science Instrument Support
ODB	Operational Database
ODM	Operational Data Message
OMT	Object Model Technique
OO	Object Oriented
OOD	Object Oriented Design
OpLAN	Operational LAN
OSI	Open System Interconnect
PACS	Polar Acquisition and Command System
PAS	Planning and Scheduling
PDB	Project Data Base
PDF	Publisher's Display Format
PDL	Program Design Language
PDR	Preliminary Design Review
PI	Principal Investigator
PI/TL	Principal Investigator/Team Leader
PID	Parameter ID
PIN	Password Identification Number
POLAR	Polar Plasma Laboratory

POP	Polar-Orbiting Platform
POSIX	Portable Operating System for Computing Environments
PSAT	Predicted Site Acquisition Table
PSTOL	PORTS System Test and Operation Language
Q/L	Quick Look
R/T	Real-Time
RAID	Redundant Array of Inexpensive Disks
RCM	Real-Time Contact Management
RDBMS	Relational Database Management System
RMA	Reliability, Maintainability, Availability
RMON	Remote Monitoring
RMS	Resource Management Subsystem
RPC	Remote Processing Computer
RTCS	Relative Time Command Sequence
RTS	Relative Time Sequence; Real-Time Server
S/C	Spacecraft
SAR	Schedule Add Requests
SCC	Spacecraft Controls Computer
SCF	Science Computing Facility
SCL	Spacecraft Command Language
SDF	Software Development Facility
SDPS	Science Data Processing Segment
SDVF	Software Development and Validation Facility
SEAS	Systems, Engineering, and Analysis Support
SEC	South Equator Crossing
SLAN	Support LAN
SMA	S-band Multiple Access
SMC	Service Management Center
SN	Space Network
SNMP	System Network Mgt Protocol
SQL	Structured Query Language
SSA	S-band Single Access
SSIM	Spacecraft Simulator
SSR	Solid State Recorder
STOL	System Test and Operations Language

T&C	Telemetry and Command
TAE	Transportable Applications Environment
TBD	To Be Determined
TBR	To Be Replaced/Resolved/Reviewed
TCP	Transmission Control Protocol
TD	Target Day
TDM	Time Division Multiplex
TDRS	Tracking and Data Relay Satellite
TDRSS	Tracking and Data Relay Satellite System
TIROS	Television Infrared Operational Satellite
TL	Team Leader
TLM	Telemetry subsystem
TMON	Telemetry Monitor
TOO	Target Of Opportunity
TOPEX	Topography Ocean Experiment
TPOCC	Transportable Payload Operations Control Center
TRMM	Tropical Rainfall Measuring Mission
TRUST	TDRSS Resource User Support Terminal
TSS	TDRSS Service Session
TSTOL	TRMM System Test and Operations Language
TW	Target Week
U.S.	United States
UAV	User Antenna View
UI	User Interface
UPS	User Planning System
US	User Station
UTC	Universal Time Code; Universal Time Coordinated
VAX	Virtual Extended Address
VMS	Virtual Memory System
W/S	Workstation
WAN	Wide Area Network
WOTS	Wallops Orbital Tracking Station
XTE	X-Ray Timing Explorer

This page intentionally left blank.

Glossary

GLOSSARY of TERMS for the Flight Operations Segment

activity	A specified amount of scheduled work that has a defined start date, takes a specific amount of time to complete, and comprises definable tasks.
analysis	Technical or mathematical evaluation based on calculation, interpolation, or other analytical methods. Analysis involves the processing of accumulated data obtained from other verification methods.
attitude data	Data that represent spacecraft orientation and onboard pointing information. Attitude data includes: • Attitude sensor data used to determine the pointing of the spacecraft axes, calibration and alignment data, Euler angles or quaternions, rates and biases, and associated parameters. • Attitude generated onboard in quaternion or Euler angle form. • Refined and routine production data related to the accuracy or knowledge of the attitude.
availability	A measure of the degree to which an item is in an operable and committable state at the start of a "mission" (a requirement to perform its function) when the "mission" is called for an unknown (random) time. (Mathematically, operational availability is defined as the mean time between failures divided by the sum of the mean time between failures and the mean down time [before restoration of function].

availability (inherent) (A_i)	The probability that, when under stated conditions in an ideal support environment without consideration for preventive action, a system will operate satisfactorily at any time. The “ideal support environment” referred to, exists when the stipulated tools, parts, skilled work force manuals, support equipment and other support items required are available. Inherent availability excludes whatever ready time, preventive maintenance downtime, supply downtime and administrative downtime may require. A_i can be expressed by the following formula: $A_i = MTBF / (MTBF + MTTR)$ Where: MTBF = Mean Time Between Failures MTTR = Mean Time To Repair
availability (operational) (A_o)	The probability that a system or equipment, when used under stated conditions in an actual operational environment, will operate satisfactorily when called upon. A_o can be expressed by the following formula: $A_o = MTBM / (MTBM + MDT + ST)$ Where: MTBM = Mean Time Between Maintenance (either corrective or preventive) MDT = Mean Maintenance Down Time where corrective, preventive administrative and logistics actions are all considered. ST = Standby Time (or switch over time)
baseline activity profile	A schedule of activities for a target week corresponding to normal instrument operations constructed by integrating long term plans (i.e., LTSP, LTIP, and long term spacecraft operations plan).
build	An assemblage of threads to produce a gradual buildup of system capabilities.
calibration	The collection of data required to perform calibration of the instrument science data, instrument engineering data, and the spacecraft engineering data. It includes pre-flight calibration measurements, in-flight calibrator measurements, calibration equation coefficients derived from calibration software routines, and ground truth data that are to be used in the data calibration processing routine.

command	Instruction for action to be carried out by a space-based instrument or spacecraft.
command and data handling (C&DH)	The spacecraft command and data handling subsystem which conveys commands to the spacecraft and research instruments, collects and formats spacecraft and instrument data, generates time and frequency references for subsystems and instruments, and collects and distributes ancillary data.
command group	A logical set of one or more commands which are not stored onboard the spacecraft and instruments for delayed execution, but are executed immediately upon reaching their destination on board. For the U.S. spacecraft, from the perspective of the EOS Operations Center (EOC), a preplanned command group is preprocessed by, and stored at, the EOC in preparation for later uplink. A real-time command group is unplanned in the sense that it is not preprocessed and stored by the EOC.
detailed activity schedules	The schedule for a spacecraft and instruments which covers up to a10 day period and is generated/updated daily based on the instrument activity listing for each of the instruments on the respective spacecraft. For a spacecraft and instrument schedule the spacecraft subsystem activity specifications needed for routine spacecraft maintenance and/or for supporting instruments activities are incorporated in the detailed activity schedule.
direct broadcast	Continuous down-link transmission of selected real-time data over a broad area (non-specific users).
EOS Data and Operations System (EDOS) production data set	Data sets generated by EDOS using raw instrument or spacecraft packets with space-to-ground transmission artifacts removed, in time order, with duplicate data removed, and with quality/ accounting (Q/A) metadata appended. Time span or number of packets encompassed in a single data set are specified by the recipient of the data. These data sets are equivalent to Level 0 data formatted with Q/A metadata. For EOS, the data sets are composed of: instrument science packets, instrument engineering packets, spacecraft housekeeping packets, or onboard ancillary packets with quality and accounting information from each individual packet and the data set itself and with essential formatting information for unambiguous identification and subsequent processing.

housekeeping data	The subset of engineering data required for mission and science operations. These include health and safety, ephemeris, and other required environmental parameters.
instrument	• A hardware system that collects scientific or operational data. • Hardware-integrated collection of one or more sensors contributing data of one type to an investigation. • An integrated collection of hardware containing one or more sensors and associated controls designed to produce data on/in an observational environment.
instrument activity deviation list	An instrument's activity deviations from an existing instrument activity list, used by the EOC for developing the detailed activity schedule.
instrument activity list	An instrument's list of activities that nominally covers seven days, used by the EOC for developing the detailed activity schedule.
instrument engineering data	subset of telemetered engineering data required for performing instrument operations and science processing
instrument microprocessor memory loads	Storage of data into the contents of the memory of an instrument’s microprocessor, if applicable. These loads could include microprocessor-stored tables, microprocessor-stored commands, or updates to microprocessor software.
instrument resource deviation list	An instrument's anticipated resource deviations from an existing resource profile, used by the EOC for establishing TDRSS contact times and building the preliminary resource schedule.
instrument resource profile	Anticipated resource needs for an instrument over a target week, used by the EOC for establishing TDRSS contact times and building the preliminary resource schedule.
instrument science data	Data produced by the science sensor(s) of an instrument, usually constituting the mission of that instrument.
long-term instrument plan (LTIP)	The plan generated by the instrument representative to the spacecraft's IWG with instrument-specific information to complement the LTSP. It is generated or updated approximately every six months and covers a period of up to approximately 5 years.

long-term science plan (LTSP)	The plan generated by the spacecraft's IWG containing guidelines, policy, and priorities for its spacecraft and instruments. The LTSP is generated or updated approximately every six months and covers a period of up to approximately five years.
long term spacecraft operations plan	Outlines anticipated spacecraft subsystem operations and maintenance, along with forecasted orbit maneuvers from the Flight Dynamics Facility, spanning a period of several months.
mean time between failure (MTBF)	The reliability result of the reciprocal of a failure rate that predicts the average number of hours that an item, assembly or piece part will operate within specific design parameters. (MTBF=1/(l) failure rate; (l) failure rate = # of failures/operating time.
mean down time (MDT)	Sum of the mean time to repair MTTR plus the average logistic delay times.
mean time between maintenance (MTBM)	The mean time between preventive maintenance (MTBPM) and mean time between corrective maintenance (MTBCM) of the ECS equipment. Each will contribute to the calculation of the MTBM and follow the relationship: 1/MTBM = 1/MTBPM + 1/ MTBCM
mean time to repair (MTTR)	The mean time required to perform corrective maintenance to restore a system/equipment to operate within design parameters.
object	Identifiable encapsulated entities providing one or more services that clients can request. Objects are created and destroyed as a result of object requests. Objects are identified by client via unique reference.
orbit data	Data that represent spacecraft locations. Orbit (or ephemeris) data include: Geodetic latitude, longitude and height above an adopted reference ellipsoid (or distance from the center of mass of the Earth); a corresponding statement about the accuracy of the position and the corresponding time of the position (including the time system); some accuracy requirements may be hundreds of meters while other may be a few centimeters.
playback data	Data that have been stored on-board the spacecraft for delayed transmission to the ground.

preliminary resource schedule	An initial integrated spacecraft schedule, derived from instrument and subsystem resource needs, that includes the network control center TDRSS contact times and nominally spans seven days.
preplanned stored command	A command issued to an instrument or subsystem to be executed at some later time. These commands will be collected and forwarded during an available uplink prior to execution.
principal investigator (PI)	An individual who is contracted to conduct a specific scientific investigation. (An instrument PI is the person designated by the EOS Program as ultimately responsible for the delivery and performance of standard products derived from an EOS instrument investigation.).
prototype	Prototypes are focused developments of some aspect of the system which may advance evolutionary change. Prototypes may be developed without anticipation of the resulting software being directly included in a formal release. Prototypes are developed on a faster time scale than the incremental and formal development track.
raw data	Data in their original packets, as received from the spacecraft and instruments, unprocessed by EDOS. • Level 0 – Raw instrument data at original resolution, time ordered, with duplicate packets removed. • Level 1A – Level 0 data, which may have been reformatted or transformed reversibly, located to a coordinate system, and packaged with needed ancillary and engineering data. • Level 1B – Radiometrically corrected and calibrated data in physical units at full instrument resolution as acquired. • Level 2 – Retrieved environmental variables (e.g., ocean wave height, soil moisture, ice concentration) at the same location and similar resolution as the Level 1 source data. • Level 3 – Data or retrieved environmental variables that have been spatially and/or temporally resampled (i.e., derived from
real-time data	Data that are acquired and transmitted immediately to the ground (as opposed to playback data). Delay is limited to the actual time required to transmit the data.
reconfiguration	A change in operational hardware, software, data bases or procedures brought about by a change in a system's objectives.

SCC-stored commands and tables	Commands and tables which are stored in the memory of the central onboard computer on the spacecraft. The execution of these commands or the result of loading these operational tables occurs sometime following their storage. The term “core-stored” applies only to the location where the items are stored on the spacecraft and instruments; core-stored commands or tables could be associated with the spacecraft or any of the instruments.
scenario	A description of the operation of the system in user’s terminology including a description of the output response for a given set of input stimuli. Scenarios are used to define operations concepts.
segment	One of the three functional subdivisions of the ECS: CSMS -- Communications and Systems Management Segment FOS -- Flight Operations Segment SDPS -- Science Data Processing Segment
sensor	A device which transmits an output signal in response to a physical input stimulus (such as radiance, sound, etc.). Science and engineering sensors are distinguished according to the stimuli to which they respond. • Sensor name: The name of the satellite sensor which was used to obtain that data.
spacecraft engineering data	The subset of engineering data from spacecraft sensor measurements and on-board computations.
spacecraft subsystems activity list	A spacecraft subsystem's list of activities that nominally covers seven days, used by the EOC for developing the detailed activity schedule.
spacecraft subsystems resource profile	Anticipated resource needs for a spacecraft subsystem over a target week, used by the EOC for establishing TDRSS contact times and building the preliminary resource schedule.
target of opportunity (TOO)	A TOO is a science event or phenomenon that cannot be fully predicted in advance, thus requiring timely system response or high-priority processing.
thread	A set of components (software, hardware, and data) and operational procedures that implement a function or set of functions.

thread, *as used in some Systems Engineering documents*	A set of components (software, hardware, and data) and operational procedures that implement a scenario, portion of a scenario, or multiple scenarios.
toolkits	Some user toolkits developed by the ECS contractor will be packaged and delivered on a schedule independent of ECS releases to facilitate science data processing software development and other development activities occurring in parallel with the ECS.